

Towards the Efficient Adaptation of Offline Physically Based Methods for Real-time Rendering

Kevin Napoli

Supervisor: Dr Keith Bugeja

Co-Supervisor: Dr Sandro Spina

May, 2025

*Submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy in Computer Science.*



L-Università ta' Malta

Faculty of Information &
Communication Technology



University of Malta Library – Electronic Thesis & Dissertations (ETD) Repository

The copyright of this thesis/dissertation belongs to the author. The author's rights in respect of this work are as defined by the Copyright Act (Chapter 415) of the Laws of Malta or as modified by any successive legislation.

Users may access this full-text thesis/dissertation and can make use of the information contained in accordance with the Copyright Act provided that the author must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the prior permission of the copyright holder.

To my family

Acknowledgements

The completion of this doctoral journey would not have been possible without the support and guidance of numerous individuals, to whom I wish to express my profound gratitude.

I extend my deepest thanks to my primary supervisor, Dr Keith Bugeja, and my co-supervisor, Dr Sandro Spina. Their indispensable guidance and expertise were the bedrock upon which this research was built. I am particularly grateful for their patience and their invaluable ability to help me identify the next research targets, especially during periods of indecision. Without their steadfast support, this thesis would not have come to fruition.

I am also indebted to my friends and colleagues who provided support throughout this process. A special mention must go to Mark, who dedicated countless hours to proofreading my papers and this thesis, and who offered invaluable moral support at all times. Adrian also deserves my sincere thanks for his willingness to proofread my work and for sharing his vast, up-to-date knowledge of the field. I am also thankful to Jennifer for her help and encouragement along the way.

My profound gratitude extends to my family. To my mum and dad, thank you for your unwavering support, for checking on me every day, and for always rooting for me. I also wish to remember my beloved grandmother, Grace, who passed away during the writing of this thesis. *Nanna*, I am eternally grateful for the love and encouragement you always gave me, for the values you instilled in us, and for the many summers I spent in your care. You played a significant role in shaping the person I am today.

Finally, and most importantly, I wish to thank my loving wife, Rachel. Her unconditional support has been my greatest source of strength. Throughout my studies, which took far longer than anticipated, she never once rushed me and continually offered her reassurance. Her own dedication is astounding, balancing two jobs and managing the housework single-handedly during the most intense periods of my research, sometimes for weeks on end. I look forward to the future where I can repay this kindness, supporting her in whatever she chooses to pursue. Rachel, this achievement is as much yours as it is mine.

Abstract

Rendering physically accurate caustics in real-time remains a persistent challenge due to their complex light interactions and high-frequency features. This thesis presents a comprehensive exploration into adapting offline physically based rendering techniques for real-time caustic synthesis on modern GPU architectures. Central to this work is CandelaDXR, a GPU-accelerated light tracer that employs novel importance sampling strategies to improve convergence speed and visual fidelity for caustics. By generating dynamic probability distribution functions conditioned on scene geometry, camera parameters, and material properties, CandelaDXR prioritises specular interactions and focuses sampling efforts on perceptually relevant regions. To support this system, two auxiliary tools were developed: Anvil, a modular visual debugging platform for rendering pipelines, and Forge, an evaluation framework designed to facilitate reproducible, cross-system comparisons. These tools provide both insight and rigour in diagnosing rendering artefacts and validating algorithmic improvements. Additionally, the thesis introduces a spectral denoising pipeline tailored to the distinct characteristics of caustic signals, demonstrating the effectiveness of Fourier, wavelet and curvelet-based transforms in preserving detail while reducing noise. Quantitative results across multiple scenes and viewpoints reveal significant performance gains, noise reduction, and perceptual improvements in CandelaDXR over baseline methods. Collectively, this work contributes a unified architecture for real-time caustic rendering, debugging, and evaluation, offering practical advances in both rendering theory and implementation for real-time physically based graphics.

List of Publications

Kevin Napoli, Keith Bugeja, Sandro Spina, Mark Magro, and Adrian De Barro. Anvil: A Tool for Visual Debugging of Rendering Pipelines. In Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2022) - GRAPP, pages 196–203. SciTePress, 2022.

Kevin Napoli, Keith Bugeja, Sandro Spina, and Mark Magro. Spectral Transforms for Caustic Denoising: A Comparative Analysis for Monte Carlo Rendering. In Advances in Computer Graphics. CGI 2024. Lecture Notes in Computer Science, vol 15338, pages 212–225. Springer, 2025.

Kevin Napoli, Keith Bugeja, and Sandro Spina. Forge: Extending Anvil for Visual Evaluation of Rendering Pipelines. In Proceedings of the 20th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - GRAPP, pages 259–266. SciTePress, 2025.

Contents

Acknowledgements	ii
Abstract	iii
List of Publications	iv
List of Figures	xiii
List of Tables	xv
List of Abbreviations	xviii
1 Introduction	1
1.1 Rendering Techniques	2
1.2 Advances in Hardware	2
1.3 Caustics	3
1.4 Motivation	4
1.5 Research Questions	6
1.6 Research Methodology	7
1.6.1 Candela	7
1.6.2 CandelaDXR - Naive	7
1.6.3 Anvil	8
1.6.4 Spectral Denoising	8
1.6.5 Forge	8
1.6.6 CandelaDXR - Importance Sampling	8
1.7 Contributions	9
1.8 Thesis Structure	10
2 Background	11
2.1 Mathematics	11
2.1.1 Solid Angle	11
2.1.2 Integration	12
2.1.3 Double Integrals	12

2.1.4	Surface Integrals	13
2.2	Radiometry	18
2.3	Surface Interaction Models	19
2.3.1	BRDF Models	21
2.4	The Rendering Equation	23
2.4.1	The Importance Equation	24
2.5	Monte Carlo	25
2.6	Path Tracing	25
2.7	Candela	26
2.8	Software Design	26
2.8.1	Software Quality Attributes	27
2.9	WebSockets	27
2.10	Spectral Transforms	28
2.11	Search	28
2.12	Image Quality Metrics	29
2.13	Denoising	31
2.14	Summary	32
3	Literature Review	33
3.1	Ray-based	33
3.2	Radiosity	36
3.3	Photon Mapping	37
3.4	Manifold-based	38
3.5	Resampling	40
3.6	Other Methods	41
3.7	Related Work	42
3.7.1	Debugging Tools for Rendering Pipelines	42
3.7.2	Caustics Rendering	44
3.7.3	Evaluation Framework	45
3.7.4	Spectral Rendering and Denoising	46
3.8	Summary	47
4	Anvil	49
4.1	Initial Methodology	49
4.2	Path Tracer Verification	50
4.3	Debugging Challenges	51
4.4	Introducing Anvil	51
4.5	Design Principles	52
4.6	Design	53

4.6.1	Reflection	54
4.6.2	Compatible Members	55
4.7	Visualisation and Analysis	55
4.7.1	OpenGL Visualisers	57
4.7.2	Example Line Visualiser	58
4.7.3	Third-party Library Considerations	59
4.8	Registry	60
4.9	Pipeline Flow	60
4.9.1	Initialisation Stage	61
4.9.2	Runtime Stage	61
4.10	Typical Usage Example	62
4.11	Evaluation	63
4.11.1	Path Tracing	63
4.11.2	Bidirectional Path Tracing	67
4.11.3	Verification Testing	69
4.12	Discussion	71
4.13	Summary	71
5	Light Tracing	73
5.1	Light Tracing	74
5.1.1	Derivation	75
5.2	Candela	77
5.2.1	Materials	77
5.2.2	Candela Path Tracer	78
5.2.3	Candela Light Tracer	80
5.2.4	Methodology	80
5.2.5	Evaluation	83
5.2.6	Discussion	91
5.3	CandelaDXR	94
5.3.1	Design	95
5.3.2	Concurrency Considerations	99
5.3.3	Naive Light Tracer Pipeline	100
5.3.4	Methodology	101
5.3.5	Evaluation	103
5.3.6	Discussion	104
5.4	Summary	107
6	Forge	108
6.1	Challenges in Light Tracing and Denoising	108

6.2	Complexity in Graphics Software Evaluation	109
6.3	Introducing Forge	109
6.4	Forge - An Anvil Evaluation Framework	110
6.5	Design	111
6.5.1	Synchronisation	111
6.5.2	WebSocket Synchronisation	112
6.6	Usage	114
6.7	Case Studies	115
6.7.1	Light and Path Tracing Verification	115
6.7.2	Spectral Denoising	117
6.8	Discussion	120
6.9	Summary	121
7	Spectral Denoising	123
7.1	Methodology	124
7.2	Experimental Setup	126
7.2.1	Scene Preparation	128
7.2.2	Transform Partitioning	128
7.2.3	Search	129
7.2.4	Denoisers	130
7.2.5	Confidence	131
7.2.6	Colour	131
7.2.7	Image Transformation	132
7.2.8	Coefficient Scene Similarity	132
7.2.9	Different Scenes	132
7.3	Evaluation	133
7.3.1	Confidence Interval	135
7.3.2	Colour Images	135
7.3.3	Image Transformation	137
7.3.4	Coefficient Scene Similarity	140
7.3.5	Different Scenes	142
7.3.6	Discussion	142
7.4	Summary	147
8	Importance	148
8.1	Caustics-Optimised Light Tracing	148
8.2	Importance Map Light Tracer	149
8.2.1	Population Methods	149
8.2.2	CDF Sampling	151

8.2.3	Interpolation of PDFs	151
8.3	Limitations	155
8.4	Methodology	155
8.5	Experimental Setup	156
8.5.1	Scenes	156
8.5.2	Image Generation	157
8.5.3	Image Comparison and Metrics	158
8.5.4	Visualisation and Analysis	158
8.5.5	Denoising	158
8.5.6	Verdict and Discussion	159
8.6	Evaluation	159
8.6.1	Variations	161
8.6.2	Denoising	181
8.7	Discussion	182
8.7.1	Denoising	186
8.8	Summary	187
9	Conclusion	189
9.1	Contributions	190
9.1.1	Anvil	190
9.1.2	Light Tracing	191
9.1.3	Forge	191
9.1.4	Spectral Denoising	192
9.1.5	Importance	193
9.2	Findings and Insight	194
9.3	Limitations and Future Work	195
9.4	Final Remarks	195
	References	205

List of Figures

Figure 1.1	Caustics formed by a glass of water	3
Figure 1.2	Glass sphere focusing light rays	4
Figure 2.1	The relationship between solid angle and polar coordinates	12
Figure 2.2	Graph of $y = x^2$. The integral in green computes $\int_0^1 x^2 dx$	13
Figure 2.3	Radiosity M on a flat surface. red is high, blue is medium, green is low and white is none	14
Figure 2.4	Surface integral mapping	14
Figure 2.5	Left: An incoming yellow ray is scattered and distributed into multiple green rays. Right: Multiple incoming green rays contribute their energy to form a single outgoing yellow ray.	20
Figure 2.6	Left: Isotropic reflection; Right: Anisotropic reflection [7]	21
Figure 3.1	Difficult case for path tracing (adapted from [7])	35
Figure 3.2	Bidirectional path tracing (adapted from [9])	36
Figure 3.3	Energy leak in photon mapping (adapted from [57])	37
Figure 4.1	Anvil debugging session. From left to right: Application framebuffer view, path view, debug view, overlay	52
Figure 4.2	Anvil architecture	54
Figure 4.3	Anvil data flow diagram	61
Figure 4.4	Rendering artefacts (fireflies) persisting after 50K samples	64
Figure 4.5	Path visualiser focusing on the anomaly	65
Figure 4.6	Path visualiser showing additional information	66
Figure 4.7	Rendering artefacts persisting after 4K samples	67
Figure 4.8	Web visualiser showing hit points at depth 2	69
Figure 4.9	Anvil debug view during exploratory analysis	70
Figure 5.1	The process of light tracing with a pinhole camera	74
Figure 5.2	Candela Material. Rays: red is incoming, green is diffuse, blue is reflection and orange is refraction.	78
Figure 5.3	The Cornell box rendered with PT and CLT at 2048 spp	84
Figure 5.4	Anvil web radiance viewer	85
Figure 5.5	The Cornell box rendered with PT and CLT at 1, 4, and 16 spp.	86

Figure 5.6	The Cornell box rendered with PT and CLT at 64 and 256 spp.	87
Figure 5.7	The glass sphere rendered with PT and CLT at 1, 4, and 16 spp	88
Figure 5.8	The glass sphere rendered with PT and CLT at 64 and 256 spp	89
Figure 5.9	Comparison of a detail from Figures 5.7 and 5.8 against ground truth using High-Dynamic-Range Visual-Difference-Predictor (HDRVDP) version 2.2.2. Q is a metric returned by HDRVDP; a value of 100 represents the best quality (perfect match) and lower values represent lower quality. The path-traced image (caustic) at 256 samples per pixel (spp) is of lower quality than light-traced (CLT) images at 16 samples.	90
Figure 5.10	The ring scene - observation of setup	91
Figure 5.11	The ring rendered with PT and CLT at 1, 4, and 16 spp	92
Figure 5.12	The ring rendered with PT and CLT at 64 and 256 spp	93
Figure 5.13	CandelaDXR Material. Rays: red is incoming, green is diffuse, blue is reflection and orange is refraction.	95
Figure 5.14	Poor sampler usage in Light Tracer. LCG (1664525, 1013904223). Artefacts can be observed as wave patterns	96
Figure 5.15	Visualisation of constrained directional light above scene	98
Figure 5.16	24-bit Range	100
Figure 5.17	16-bit Range	100
Figure 5.18	Caustics Buffers at 16 spp	105
Figure 5.19	HDRVDP3	106
Figure 6.1	JSON configuration for a general evaluation system	111
Figure 6.2	General process diagram for one evaluation. (+) is add, (-) is remove.	112
Figure 6.3	Multiple Forge instances communicating through the synchronisation specification. The synchronisation block (dashed rectangle) depicts the synchronisation process; the synchronisation systems are implicitly represented between System sets A and B, and encompassed in System set C.	113
Figure 6.4	Left: Path traced, Right: Light traced	116
Figure 6.5	Light and path tracing flow diagram.	117
Figure 6.6	Spectral flow diagram.	119
Figure 6.7	Spectral animation MSE score (lower is better).	119
Figure 6.8	Frame 8. Left: noisy, right: denoised.	120
Figure 7.1	Collection of various images used in the study (image brightness enhanced for illustrative purposes). Each image is divided into four quadrants. Top-left: Ground truth image. Top-right: Noisy input. Bottom-left: Result from the OptiX denoiser. Bottom-right: Best denoised candidate from traditional denoisers.	127

Figure 7.2	Coefficient filters for DICE. Left: Fourier. Centre: Wavelet. Right: Curvelet.	129
Figure 7.3	Thresholding functions ($t = 0.5$). Black is hard, red is soft, purple is garrote, green is multiplication.	130
Figure 7.4	Left: ReLAX denoised. Right: Curvelet denoised.	133
Figure 7.5	Dice scene denoised	136
Figure 7.6	RGB denoised	139
Figure 7.7	Image translations applied to OUTER (aggregate coefficients)	141
Figure 7.8	Denoising scenes with different coefficients.	143
Figure 7.9	Clustered speckle noise in DICE with VDP3.	145
Figure 7.10	Left: Reference. Middle: OptiX. Right: Curvelet.	145
Figure 8.1	CDF plane inside the hemisphere subtended by a light source	152
Figure 8.2	probability distribution function (PDF) interpolation example with $w = 0.7$	155
Figure 8.3	Spike in MSE with PT with raw render	160
Figure 8.4	EGG_CLOSEUP buffers, variations and plots	162
Figure 8.5	Radiance Buffers	163
Figure 8.6	Caustics Buffers	164
Figure 8.7	Cropped Radiance Buffers	165
Figure 8.8	Blurred Radiance Buffers	166
Figure 8.9	CUP renders	167
Figure 8.10	CUPS renders	168
Figure 8.11	DICE renders	169
Figure 8.12	REFLECT renders	170
Figure 8.13	OUTER renders	171
Figure 8.14	TORUS renders	172
Figure 8.15	EGG renders	173
Figure 8.16	EGG_CLOSEUP renders	174
Figure 8.17	WATER renders	175
Figure 8.18	Aggregate	176
Figure 8.19	Aggregate - Blurred - 2	177
Figure 8.20	Aggregate - Downsampled 25%	178
Figure 8.21	Aggregate - Cropped	179
Figure 8.22	Collection of various images used in the study (image brightness enhanced for illustrative purposes). Each image is divided into four quadrants. Top-left: Ground truth image. Top-right: Noisy input. Bottom-left: Result from the OptiX denoiser. Bottom-right: Best denoised candidate from traditional denoisers.	180

Figure 8.23 Metric scores for the noisy images.	181
Figure 8.24 OptiX denoised. Non-importance vs. importance.	183
Figure 8.25 Spectral denoised. Non-importance vs. importance.	184
Figure 8.26 Clustered speckle noise in $DICE$ with VDP3.	187
Figure 8.27 Left: Reference. Middle: OptiX. Right: Curvelet.	187

List of Tables

Table 2.1	Radiometric quantities	19
Table 4.1	Registrations	56
Table 4.2	Registrations of private data members	56
Table 5.1	Variable Definitions for all Algorithms	79
Table 5.2	Variable Definitions Specific to Algorithm 2	83
Table 5.3	Variable Definitions Specific to Algorithm 3	101
Table 5.4	LT Shader performance per frame (approximate in ms) per scene. Hardware: Intel i7-8700K, NVIDIA GeForce RTX 2080 Ti	104
Table 6.1	Cornell Box Shiny comparing light and path traced scores. TM = ACES tone mapping. FLIP and MSE, lower is better.	116
Table 7.1	Metric scores for the noisy images.	133
Table 7.2	Accumulating and non-accumulating scores. Best score for each scene- metric pair shown in bold. Top: ReLAX. Bottom: ReBLUR.	134
Table 7.3	Best scores from multiple runs (detailed); gp is Bayesian optimisation. T_s is search time, T_d is denoising time. ¹	134
Table 7.4	Aggregate metrics with all metrics computed.	134
Table 7.5	Aggregate metrics. T_s is search time, T_d is denoising time. ²	134
Table 7.6	OptiX accumulating scores.	137
Table 7.7	Best scores from multiple runs.	137
Table 7.8	FLIP metric comparison between the best curvelet-based denoiser and the OptiX denoiser. Figures in bold indicate the instances where the curvelet- based denoiser obtained a better metric value.	137
Table 7.9	Aggregate metrics 95% confidence interval. Values in bold represent scores outside the interval.	137
Table 7.10	Metric scores for the noisy coloured images.	138
Table 7.11	Metric scores for the denoised coloured images.	138
Table 7.12	Aggregate scores for the denoised coloured images, including denois- ing time.	138
Table 7.13	Aggregate scores cosine similarity matrix.	142

Table 7.14	Metric scores for the denoised scenes with different coefficients. A-B means scene A denoised with coefficients derived from scene B.	143
Table 7.15	Ratios for DICE. Higher (bold) is better.	145
Table 8.1	Variable Definitions for Algorithms 6 and 7	152
Table 8.2	Scene setup configuration	157
Table 8.3	Shader descriptions	157
Table 8.4	Samples per Pixel for Each Image Number	158
Table 8.5	Shader performance per frame (approximate in ms) per scene. Hardware: Intel i7-8700K, NVIDIA GeForce RTX 2080 Ti	161
Table 8.6	Accumulating and non-accumulating scores. Best score for each scene-metric pair shown in bold. Top: ReLAX. Bottom: ReBLUR.	181
Table 8.7	Best scores from multiple runs (detailed); gp is Bayesian optimisation. T_s is search time, T_d is denoising time. ²	181
Table 8.8	Aggregate metrics with all metrics computed.	182
Table 8.9	Aggregate metrics. T_s is search time, T_d is denoising time. ¹	182
Table 8.10	OptiX accumulating scores.	185
Table 8.11	Best scores from multiple runs.	185

List of Abbreviations

AABB	Axis-Aligned Bounding Box
AAPS	Adaptive Anisotropic Photon Scattering
ACES	Academy Colour Encoding System
API	Application Programming Interface
BFGS	Broyden–Fletcher–Goldfarb–Shanno
BPT	Bidirectional Path Tracing
BRDF	Bidirectional Reflectance Distribution Function
BSDF	Bidirectional Scattering Distribution Function
BSSDF	Bidirectional Scattering-Surface Distribution Function
BTDF	Bidirectional Transmittance Distribution Function
BVH	Bounding Volume Hierarchy
CDF	Cumulative Density Function
CG	Computer Graphics
CLT	Candela Light Tracer
CNN	Convolutional Neural Network
COBYLA	Constrained Optimisation By Linear Approximations
COLT	Caustics-Optimised Light Tracing
DFT	Discrete Fourier Transform
DWT	Discrete Wavelet Transform
DXR	DirectX Raytracing
ECA	Entity-Component-Attribute
ECS	Entity Component System
EXR	Extended Dynamic Range
FFT	Fast Fourier Transform
FHD	Full High Definition
FPS	frames per second
GI	Global Illumination
GPGPU	General-Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
GRIS	Generalised Resampled Importance Sampling

GUI	Graphical User Interface
GUID	Globally Unique Identifier
HDR	High Dynamic Range
HDRVDP	High-Dynamic-Range Visual-Difference-Predictor
HDRVDP3	High-Dynamic-Range Visual-Difference-Predictor v3
IETF	Internet Engineering Task Force
L-BFGS-B	Limited-Memory Broyden-Fletcher-Goldfarb-Shanno with Boundaries
LCG	Linear Congruential Generator
LDR	Low Dynamic Range
LPIPS	Learned Perceptual Image Patch Similarity
LT	Light Tracing
MLT	Metropolis Light Transport
MMO	Massively Multiplayer Online
MNEE	Manifold Next Event Estimation
MSE	Mean Squared Error
NEE	Next Event Estimation
OOP	Object-Oriented Programming
PBR	Physically Based Rendering
PCG	Permuted Congruential Generator
PCM	Procedural Caustic Mesh
PDF	Probability Distribution Function
PDS	Photon Difference Scattering
PM	Photon Mapping
PRNG	Pseudo-Random Number Generator
PSNR	Peak Signal-to-Noise Ratio
PT	Path Tracing
RGWC	Ray-Guided Water Caustics
RIS	Resampled Importance Sampling
SID	Spectral Image Denoising

SLSQP	Sequential Least Squares Programming
SMS	Specular Manifold Sampling
spp	Samples Per Pixel
SSIM	Structural Similarity Index
SVGF	Spatiotemporal Variance-Guided Filtering
TNC	Truncated Newton Conjugate Gradient
VDP3	High-Dynamic-Range Visual-Difference-Predictor v3
VR	Virtual Reality

1 Introduction

Image synthesis refers to the generation of an image based on a detailed description of a scene. This description typically includes information such as the vertices associated with an object and the material properties of its composition. A renderer, which is a specialised software application, performs this process of image synthesis, commonly referred to as rendering. Although rendering is frequently associated with video game development, its applications extend significantly beyond this domain, encompassing areas such as prototyping, serious games, and archaeological reconstruction.

Rendering can be classified into two categories based on temporal constraints: *offline* and *real time*. Offline rendering is employed in applications where time is not a critical factor, such as the production of animated films. In contrast, real-time rendering is utilised in scenarios where images must be generated within a strict time frame. For instance, a single frame in an animated film may require several hours or even days to produce¹, whereas a frame in a video game must be rendered within 16.7 milliseconds to maintain a display rate of 60 frames per second (FPS). This stringent time constraint encompasses not only the rendering process itself but also the execution of logic, such as object animation and the computation of physical interactions (e.g., gravity and collisions). Consequently, the actual time allocated for rendering is less than 16.7 milliseconds. Furthermore, virtual reality (VR) applications impose an even stricter limit of under 14 milliseconds (72 Hz) to mitigate the risk of motion sickness.

Rendering may alternatively be classified according to the visual style produced by the renderer. Renderers are capable of generating diverse image types, ranging from stylised cartoon imagery [1] to photorealistic representations (artistic interpretations of real-world scenes). Physically based rendering (PBR) specifically generates images through the computational simulation of light transport, with the primary objective of attaining photorealism.

Physically based renderers, whether operating in offline or real-time modes, aim to approximate solutions to the *rendering equation* [2]. This fundamental equation, grounded in the principle of energy conservation, mathematically describes light transport and energy exchange between surfaces within a virtual environment. Formulated as a Fredholm integral equation of the second kind, it lacks an analytical solution and consequently requires numerical approximation methods for its resolution.

$$L_o = L_e + \int_{\Omega} f_r L_i \cos(\theta_i) d\omega_i \quad (1.1)$$

Equation 1.1 presents the rendering equation restricted to reflection phenomena. It mathematically defines the radiance L_o emitted in a specific direction, accounting for

¹<https://renderman.pixar.com/stories/the-world-of-coco>

both incoming illumination L_i from hemispherical directions at a given point and any self-emitted light L_e . The bidirectional reflectance distribution function (BRDF), denoted f_r , quantifies the proportion of incident light that is reflected. Conceptually, this formulation represents a recursive integral equation where the recursion depth corresponds to light transport interactions. The equation can alternatively be interpreted as a Liouville–Neumann series expansion, with each successive term modelling an additional light bounce. Renderers designed to produce physically accurate results seek computationally feasible approximations of this fundamental equation.

1.1 Rendering Techniques

Numerous rendering techniques exist, with *rasterisation* being the most widely adopted. This method excels in real-time applications, making it the standard choice for video game development. Empirical evidence demonstrates that rasterisation has progressively achieved increasingly realistic results, including photorealistic output. The efficiency of rasterisation is significantly enhanced by dedicated hardware, specifically graphics processing units (GPUs).

Ray tracing represents an alternative rendering technique that enables the achievement of both photorealistic and physically accurate results. This approach simulates the propagation and interaction of light rays with scene geometry, closely modelling the behaviour of electromagnetic radiation. Implementations typically incorporate radiometric principles, producing images that may be indistinguishable from real-world photographs. The computational complexity of ray tracing has historically limited its application to offline rendering scenarios, such as feature film production. However, recent technological advancements have facilitated its adoption in interactive and real-time systems.

1.2 Advances in Hardware

Recent hardware developments have enabled real-time ray tracing through dedicated ray tracing cores. NVIDIA's 20-series GPUs represented the first consumer-grade graphics processors to incorporate hardware-accelerated ray tracing. This innovation was subsequently followed by AMD's Radeon 6000 series. These advancements have facilitated the adoption of *hybrid rendering*, where rasterisation and ray tracing are combined to render virtual scenes. Despite these improvements, certain optical phenomena remain challenging to render in real-time, such as caustics.

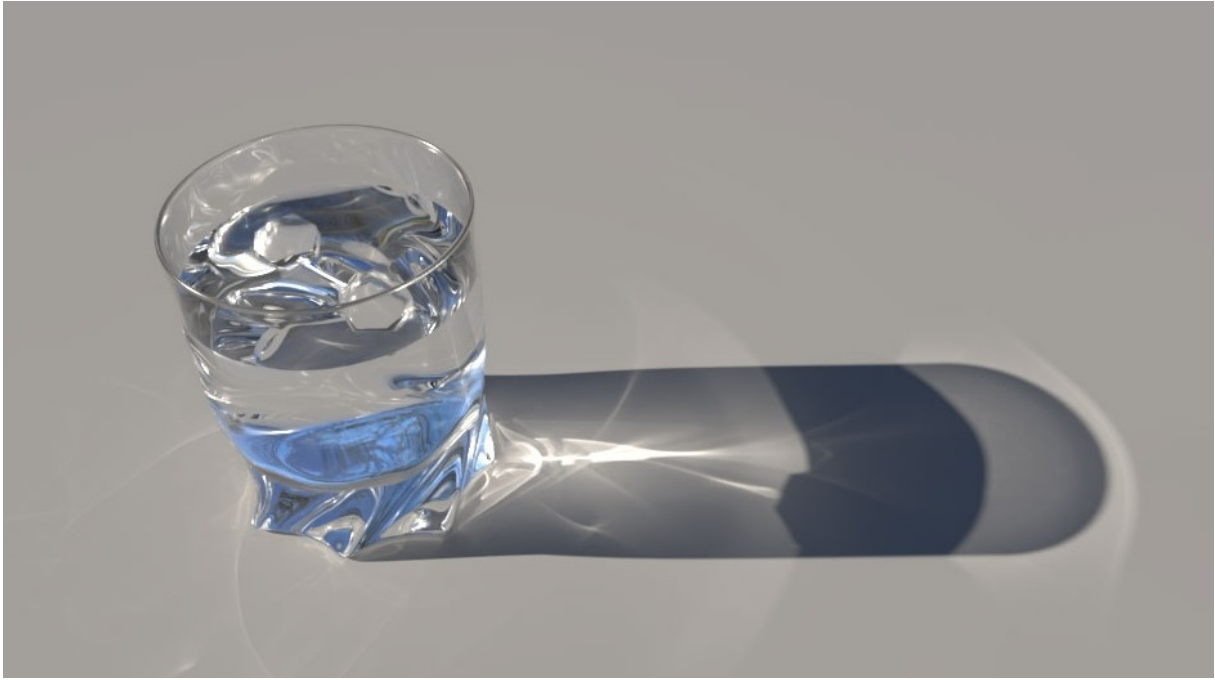


Figure 1.1 Caustics formed by a glass of water

1.3 Caustics

Caustics constitute complex optical phenomena that manifest as concentrated light patterns on diffuse surfaces when illumination undergoes reflection or refraction through transparent media. As demonstrated in Figure 1.1², this effect is exemplified by light transmission through a water-filled glass, where incident rays experience multiple refractive interactions governed by Snell’s law. The light initially refracts at the air-glass interface, undergoes further refraction through the water, and finally refracts again when exiting through the glass-air boundary. These successive directional changes of light propagation create the characteristic bright caustic patterns that present significant computational challenges for real-time rendering systems, particularly due to the need for accurate physical simulation of multi-stage light transport within strict temporal constraints.

The rendering of caustics presents significant computational challenges due to the difficulty in identifying the contributing light paths. These optical phenomena often result from light rays originating within extremely narrow angular ranges that prove problematic for conventional sampling approaches. As illustrated in Figure 1.2, the underlying physical process involves light rays converging to a minute focal region before diverging to illuminate a surface. In this configuration, any observation point along the illuminated surface (represented by the brown line) must effectively sample this vanishingly small convergence region, requiring an impractically narrow field of view.

²<https://forum.d5render.com/t/glass-shadows/111>

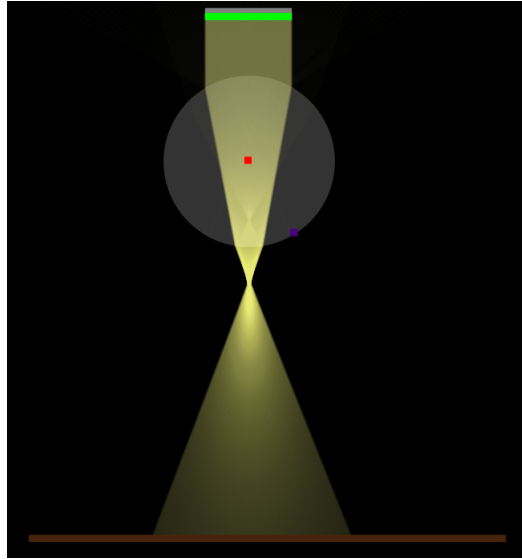


Figure 1.2 Glass sphere focusing light rays

This simplified example demonstrates that even elementary caustic formations can pose substantial rendering difficulties, highlighting the inherent complexity of accurately simulating such light transport phenomena.

Unidirectional rendering techniques, including path tracing and light tracing (forward path tracing), typically exhibit suboptimal performance when rendering caustics. Alternative algorithms such as bidirectional path tracing (BPT) [3] and Metropolis Light Transport (MLT) [4] demonstrate superior performance in handling such cases. MLT employs an adaptive sampling strategy, where high-energy ray directions are identified and their neighbourhoods are explored through small perturbations. While these advanced algorithms improve caustics rendering dramatically, their computational complexity renders them unsuitable for real-time applications. The fundamental limitation stems from the inherent trade-off between convergence speed and computational cost, where MLT's improved convergence comes at the expense of increased per-iteration processing time. Consequently, these methods remain primarily confined to offline rendering scenarios where strict real-time constraints are not imposed.

1.4 Motivation

Achieving both real-time performance and unbiased rendering presents a significant challenge. On the one hand, offline rendering techniques offer a diverse array of methods capable of producing unbiased images, yet these approaches typically necessitate substantial computational time. On the other hand, advancements in hardware have facilitated the integration of ray tracing into real-time rendering pipelines. However, this development restricts the range of viable rendering algorithms, thereby limiting the

scope of techniques that can be effectively employed. Additionally, rendering caustics further limits this selection.

A critical step towards this objective would be to identify an unbiased offline rendering technique capable of operating in real-time and executing efficiently on the GPU. Prior to implementation of such an algorithm on the GPU, a feasibility study should be conducted to assess whether the technique can effectively generate caustics and achieve acceptable convergence with a limited number of samples per pixel.

The architecture of GPUs differs fundamentally from that of CPUs, and mapping an algorithm onto a GPU is not a straightforward, one-to-one process. GPUs are designed to execute threads in lock-step wherever possible to avoid significant performance penalties. Additionally, the distinct memory hierarchy of GPUs must be carefully managed, as overlooking its intricacies can lead to substantial performance degradation. These challenges are compounded when implementing rendering algorithms, particularly those employing stochastic methods to generate images. Path tracing, for instance, is a stochastic rendering technique that relies heavily on sampling. Such algorithms depend on random number generation, with actions determined by the outcomes of these random processes. Debugging stochastic renderers is inherently complex due to their non-deterministic nature, and this complexity is exacerbated when implementing them on GPUs, where even basic debugging tools, such as print statements, are often unavailable. Furthermore, diagnosing issues related to complex light phenomena, such as caustics, becomes even more demanding.

While tools exist to assist in debugging and analysing GPU-implemented algorithms, they tend to operate at a low level of abstraction. For example, NVIDIA Nsight³ enables developers to debug frames by inspecting textures, instrumenting the rendering pipeline, and, in some cases, stepping through shader code. However, visualising rendering problems solely through numerical analysis remains a significant challenge. Consider a scenario where one attempts to verify the correctness of a ray bounce event. With only numerical data available, one would need to manually reconstruct the origin and direction vectors and visualise them using a third-party tool such as Blender. Although this approach is technically feasible, it is impractical and time-consuming, highlighting the need for more intuitive debugging methodologies.

Given the stringent time constraints inherent in real-time rendering, the incorporation of denoising techniques is likely essential. However, the specific application of denoising to caustics remains an underexplored area of research. This distinction is particularly significant, as caustic signals exhibit unique characteristics, typically manifesting as high-frequency, curve-like structures. Consequently, further investigation is warranted to develop denoising methods tailored to the specific challenges posed by caustic rendering.

³<https://developer.nvidia.com/nsight-graphics>

Given that denoising techniques for diffuse components have been extensively studied, it could be advantageous to focus efforts on the caustics buffer. To explore this, the signal could be separated into distinct components: diffuse, specular, and caustics. One potential avenue for investigation could involve denoising at the spectral level, with a simple approach utilising Fourier transforms to filter specific frequencies. However, a key challenge would lie in mathematically determining the necessary coefficients for such filtering. This would require a search through the parameter space to identify optimal values. While such approaches may not be feasible for real-time applications due to their computational complexity, pre-computed coefficients could potentially be reused, reducing computational overhead. To effectively explore the search space and evaluate potential methods, the development of a robust evaluation framework is likely necessary. Nevertheless, further research would be required to assess the viability and effectiveness of such a strategy.

Depending on the denoising results, adjustments could be made to refine the approach. A critical consideration is whether improvements can be achieved at the input level, as a higher-quality input signal should, in theory, produce a better denoised image. However, any such improvements should be carefully balanced to ensure they have a minimal impact on render time, maintaining the feasibility of real-time applications.

1.5 Research Questions

The focus of this research is on caustics, with a particular emphasis on exploring and adapting established offline rendering methods for real-time applications. Modern hardware, such as accelerated ray tracing, will play a pivotal role in this endeavour. Selection criteria will be employed to determine which offline algorithms are best suited for adaptation to GPU architectures. These criteria will take into account GPU hardware features, including caches and various memory types, as well as other details that can optimise hardware performance. While performance is of paramount importance, we also strive to ensure the correctness of the implemented methods. Achieving plausibility is merely a preliminary step; our goal is to ensure that the radiometric quantities of synthesised images, prior to post-processing, are both correct and unbiased. This can be particularly challenging, especially on lower-end hardware. Additionally, we aim to investigate post-processing techniques specifically tailored for caustics, with the objective of minimising bias in the caustics signal. Our research questions therefore are:

- To what extent can offline caustics synthesis methods be adapted for real-time use?
 - How would the methods fit into the rendering pipeline?

- What would the impact on performance be?
- Would these method be composable, and if so, to which extent?
- How well would these methods translate to general-purpose graphics processing unit (GPGPU)/rasterisation hardware?
- How can bias in the post-processing stage be minimised?
 - To what extent can caustics signals be denoised?
 - Is spectral denoising effective for such signals?

1.6 Research Methodology

Our research centres on the adaptation of offline caustics synthesis techniques for real-time applications, utilising modern hardware such as GPUs with accelerated ray-tracing capabilities. The real-time generation of caustics is a field under active exploration, particularly since the advent of hardware-accelerated ray tracing. We examine methods that harness GPU-accelerated rendering to achieve real-time caustic synthesis, thereby contributing to the growing body of knowledge in this domain.

1.6.1 Candela

Candela is a CPU-based renderer designed to facilitate the investigation of path tracing and light tracing techniques. In this work, a feasibility study is conducted to determine which of the simplest ray tracing methods (path tracing or light tracing) converges caustics more rapidly. Light tracing is examined in detail, and results based on image-similarity metrics demonstrate its superior performance in caustics generation. Furthermore, this study led to the development of Anvil, a tool for the visual debugging of rendering pipelines. Anvil was utilised to debug bidirectional path tracing during the evaluation of this technique as part of a preliminary feasibility study.

1.6.2 CandelaDXR - Naive

This work embodies the findings of the preceding chapter (Candela), implementing light tracing on the GPU using DirectX Raytracing (DXR) technology. It delves into the complexities of mapping the CPU-based algorithm to the GPU, outlining various design decisions that were critical to achieving this transition. Throughout this process, Anvil played an indispensable role in debugging issues related to fireflies, ray bounces, and energy loss in real-time rendering.

1.6.3 Anvil

This work examines Anvil, a tool specifically designed for the visual debugging of rendering pipelines. Anvil is engineered to facilitate seamless integration into applications, enabling users to debug rendering processes with ease. It identifies common data structures represented as atoms and molecules, which serve as the foundation for driving visualisers. Users can create their own custom visualisers when predefined options are unavailable, and these visualisers are pluggable, allowing them to be shared and utilised within the broader community. This flexibility enhances Anvil’s utility as a collaborative and extensible debugging tool.

1.6.4 Spectral Denoising

Spectral denoising explores the efficacy of spectral transforms in denoising caustics within physically-based Monte Carlo rendering, where preserving high-frequency details, such as contours in caustics, is essential. This work provides a comparative analysis of multiple spectral algorithms, NVIDIA’s OptiX AI-accelerated denoiser, ReBLUR, and ReLAX. Additionally, search algorithms such as Powell and Constrained Optimisation By Linear Approximations (COBYLA) are investigated to determine which method yields the most optimal coefficients. The extensive parameter space required for this analysis motivated the development of Forge, an evaluation framework designed to generate and validate the results systematically.

1.6.5 Forge

Forge, an Anvil extension, streamlines evaluation in computer graphics by ensuring consistency, reducing errors, and improving reproducibility. It decouples evaluation logic from rendering engines for seamless comparisons and supports decentralised evaluations across diverse environments. Adaptable for both small experiments and large-scale rendering studies, Forge verifies techniques, compares performance, and aids algorithm development, offering a robust solution for reliable research.

1.6.6 CandelaDXR - Importance Sampling

This work builds upon CandelaDXR, enhancing the light tracer implementation through the addition of importance sampling. The motivation behind this improvement is to provide the denoiser with a cleaner input signal. Two optimisation techniques are introduced: one inspired by next event estimation (NEE) and another utilising importance maps. The study outlines these techniques, their implementation on the GPU, and evaluates their effectiveness using eight distinct viewpoints across five different scenes. A

variant of importance maps is found to outperform the other approach across multiple scenes, demonstrating its general effectiveness.

1.7 Contributions

This thesis presents several key contributions to the field of real-time physically based rendering, with a specific focus on the efficient generation of caustics:

- **Candela and CandelaDXR:** A novel GPU-based light tracing framework was developed, demonstrating that light tracing, traditionally limited to offline rendering, can be adapted for real-time caustics synthesis using modern hardware acceleration (DXR). The implementation includes both naive and importance-sampled variants, showcasing significant improvements in convergence and quality.
- **Anvil:** A modular, visual debugging tool was introduced to facilitate the inspection and validation of complex rendering behaviours, particularly in stochastic environments. Its atom-molecule abstraction enables extensible visualisation of internal rendering states, aiding development and error diagnosis.
- **Forge:** An evaluation framework built atop Anvil, Forge enables systematic, reproducible benchmarking and cross-engine comparisons of rendering methods. It supports decentralised testing scenarios and contributes to standardised evaluation practices in rendering research.
- **Spectral Denoising for Caustics:** The thesis investigates denoising techniques tailored specifically to caustic signals. A comprehensive comparison of spectral transforms and AI-based denoisers highlights the effectiveness of curvelet-based methods for high-frequency detail preservation, particularly in challenging lighting scenarios.
- **Importance Sampling Strategies:** A GPU-compatible importance sampling technique was developed, guided by dynamic probability distributions informed by scene materials and camera parameters. This method significantly improves sampling efficiency and output quality, particularly when rendering reflective and refractive surfaces.

Together, these contributions advance the feasibility of real-time caustic rendering by addressing the algorithmic, diagnostic, and evaluative challenges that have traditionally constrained this domain.

1.8 Thesis Structure

Chapter 2: Background introduces fundamental concepts in rendering, including mathematical foundations (solid angles and integration), radiometry (light measurement), surface interaction models (BRDFs), the rendering equation, Monte Carlo methods, path tracing, and software design principles like Entity Component System (ECS).

Chapter 3: Literature Review covers the evolution of ray-based rendering techniques, from early ray casting to modern methods like path tracing, photon mapping, and specular manifold sampling, along with debugging tools and frameworks for evaluating rendering systems.

Chapter 4: Anvil presents a visual debugging framework developed to address challenges in GPU-based physically based rendering, offering efficiency, extensibility, and flexibility through atom-molecule abstractions and real-time analysis tools.

Chapter 5: Light Tracing demonstrates superior efficiency for rendering caustics compared to path tracing (PT), leveraging GPU acceleration and unbiased sampling while overcoming challenges through a CPU reference implementation, Anvil debugging, and optimised data structures.

Chapter 6: Forge describes an extensible evaluation framework for computer graphics pipelines, addressing challenges such as noise reduction in light tracing and enabling reproducible, decentralised evaluations through modular design and WebSocket-based synchronisation.

Chapter 7: Spectral Denoising examines Fourier, wavelet, and curvelet techniques for caustic denoising in Monte Carlo rendering, comparing them with conventional and AI-based approaches through quantitative metrics.

Chapter 8: Importance explores improved light tracing techniques using importance sampling to enhance caustic rendering at low sample rates, evaluates performance using various metrics, and analysis denoising results.

Chapter 9: Conclusion highlights the key findings of this research, consolidates the principal contributions, evaluates the limitations of the proposed approaches, and identifies promising directions for future investigation.

2 Background

This chapter establishes the foundational knowledge required to understand the subsequent developments in this thesis. We begin by reviewing the essential mathematical tools, including solid angles and various forms of integration, which are fundamental to describing and computing light transport. The core principles of radiometry are then introduced to define the physical quantities of light that a renderer must simulate. Following this, we explore models of surface interaction, such as the BRDF, and present the Rendering Equation, which formally describes the problem of light propagation in a scene. To address the computational challenges of solving this equation, we cover key statistical methods like Monte Carlo integration and the path tracing algorithm. Finally, we survey relevant software engineering concepts, spectral transforms, search algorithms, image quality metrics, and modern denoising techniques that are instrumental in achieving practical, high-quality rendering.

2.1 Mathematics

A solid mathematical foundation is crucial for the accurate formulation and computation of light transport. This section revisits key concepts from calculus and geometry, with a particular emphasis on integration techniques and coordinate transformations. These tools are indispensable for solving the high-dimensional integrals that arise in rendering, such as the calculation of irradiance over a surface or the evaluation of the rendering equation.

2.1.1 Solid Angle

In geometry, a solid angle quantifies a region on the surface of a unit sphere and is measured in steradians (denoted as Ω or sr). It is a useful measure for defining the field of view from a specific point. A solid angle can range from 0 to 4π steradians, with 4π representing the total surface area of a unit sphere. One steradian corresponds to approximately 8% of the sphere's total surface area. Conceptually, a solid angle is to a sphere what a planar angle is to a circle (see Figure 2.1).

In rendering, the solid angle is used to describe directional quantities, such as radiance and intensity, which represent energy transfer along a specific direction. The differential solid angle defines a unique direction from the centre of a unit sphere. Its relationship with the polar angle θ and the azimuthal angle φ is given by $d\omega = \sin \theta \, d\theta \, d\varphi$. The derivation of this result is provided in an example within Section 2.1.4.

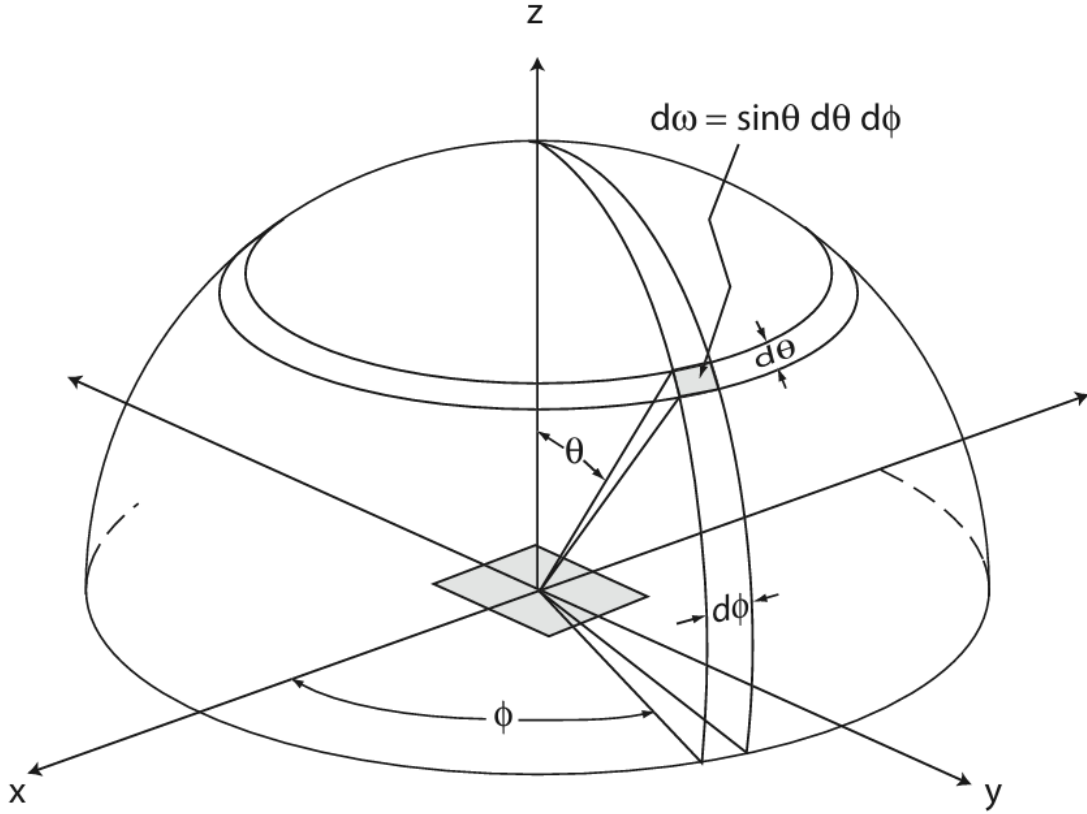


Figure 2.1 The relationship between solid angle and polar coordinates [5]

2.1.2 Integration

Integration is a well-known mathematical operation. At a high level, it can be conceptualised as the infinite sum of infinitesimal changes in a variable. The integral's definition via bounded Riemann summation is given below:

$$\int_a^b f(x)dx = \lim_{N \rightarrow \infty} \sum_{n=1}^N f(a + n\Delta x)\Delta x, \Delta x = \frac{b-a}{N} \quad (2.1)$$

Integrating a one-dimensional function yields the total area under its curve. Figure 2.2 shows the graph of $y = x^2$. This area is given by the following expression:

$$\int_0^1 x^2 dx = \left[\frac{x^3}{3} \right]_0^1 = \frac{1}{3} \quad (2.2)$$

In the context of rendering, integration can be used to convert between related quantities, such as from radiance to intensity.

2.1.3 Double Integrals

Double integrals are used to integrate two-dimensional functions. Consider a function $h(x, y)$ that returns a height value for given inputs x and y . The double integral of h

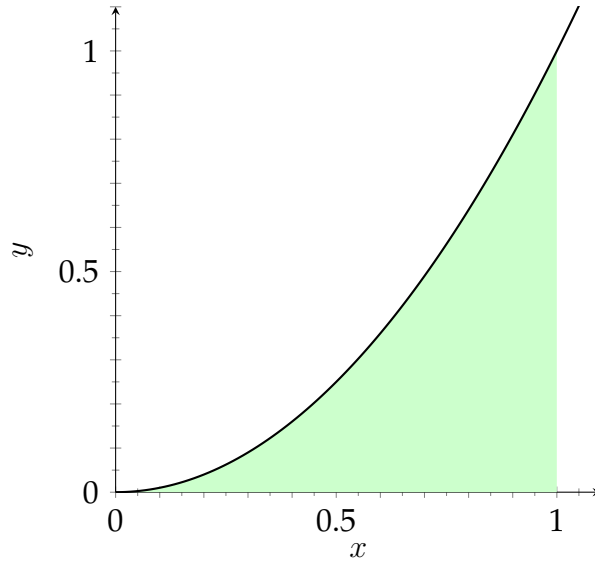


Figure 2.2 Graph of $y = x^2$. The integral in green computes $\int_0^1 x^2 dx$

yields the volume under the surface it defines. For example:

$$\int_c^d \int_a^b h(x, y) dx dy \quad (2.3)$$

A common notation for a double integral is:

$$\int_A h(x, y) dA, dA = dx dy \quad (2.4)$$

2.1.4 Surface Integrals

A surface integral generalises the concept of a double integral to integration over curved surfaces. In fact, a double integral can itself be considered a surface integral over a flat plane. For instance, consider calculating the total power emitted by a surface. Given a radiosity function $M(x, y)$ with units $\text{W} \cdot \text{m}^{-2}$, the total emanating power is obtained by integrating M over all surface elements dA :

$$P = \int_A M dA.$$

Figure 2.3 illustrates this concept, showing M discretised and multiplied by sixteen infinitesimal surface elements dA .

Computing the integral above yields the total power emanating from the surface. Surface integrals enable integration over any surface, not merely simple rectangular domains. Given a suitable parametrisation, a complex integral can be substantially simplified. The general conversion formula is given in Equation 2.5, which maps surface S onto surface T . In this process, the variables x and y are expressed as functions of s

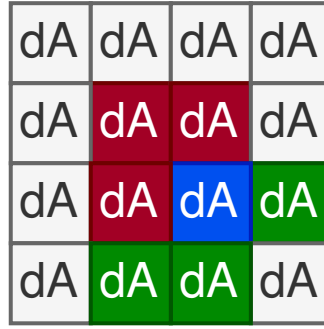


Figure 2.3 Radiosity M on a flat surface. red is high, blue is medium, green is low and white is none

and t , allowing integration with respect to s and t . The term $\frac{\partial(x,y)}{\partial(s,t)}$ denotes the Jacobian determinant, shown explicitly in Equation 2.6.

$$\int_S f(x,y) dS = \int \int_T f(x(s,t), y(s,t)) \left| \det \left[\frac{\partial(x,y)}{\partial(s,t)} \right] \right| ds dt \quad (2.5)$$

$$\frac{\partial(x,y)}{\partial(s,t)} = \begin{bmatrix} \frac{\partial x}{\partial s} & \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial s} & \frac{\partial y}{\partial t} \end{bmatrix} \quad (2.6)$$

It is also worth noting that Equation 2.5 can be expressed in vector form. Given a position vector $\vec{r} = x\vec{i} + y\vec{j}$ and a function $f(x,y)$, the relationship becomes:

$$\int_S f(x,y) dS = \int \int_T f(x(s,t), y(s,t)) \left\| \frac{\partial \vec{r}}{\partial s} \times \frac{\partial \vec{r}}{\partial t} \right\| ds dt \quad (2.7)$$

An illustration of this parametrisation process is provided in Figure 2.4. In the figure, a surface is parameterised such that infinitesimal changes ds and dt in the do-

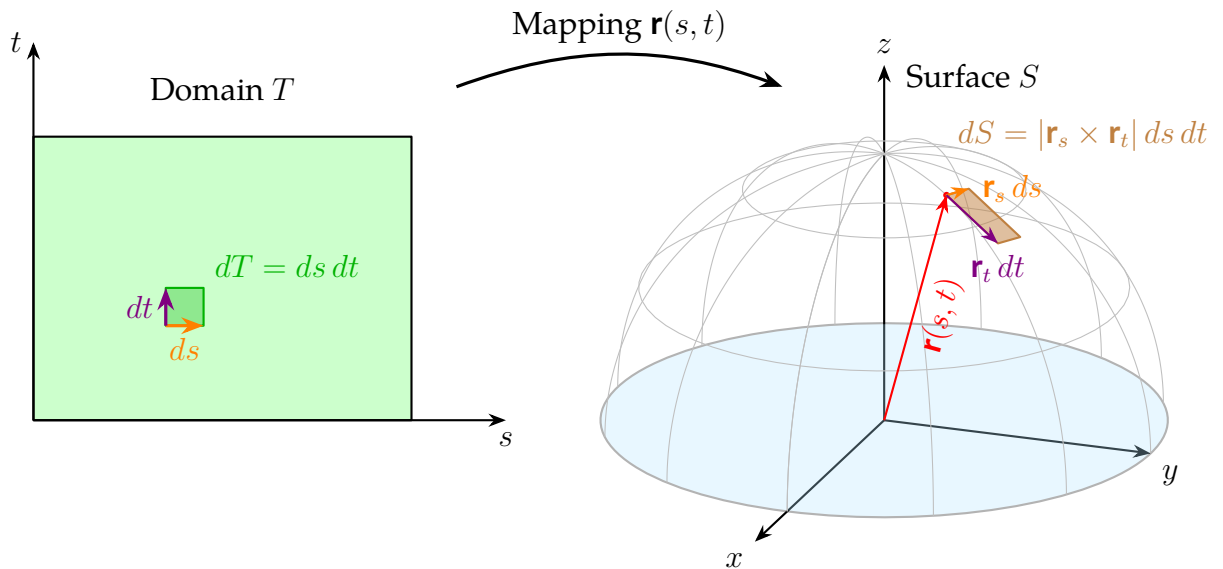


Figure 2.4 Surface integral mapping

main T map to corresponding vectors $\mathbf{r}_s ds$ and $\mathbf{r}_t dt$ on the surface, where $\mathbf{r}_s$ is $\frac{\partial \vec{r}}{\partial s}$, and $\mathbf{r}_t$ is $\frac{\partial \vec{r}}{\partial t}$. Consequently, the differential area element is scaled by the magnitude of the Jacobian, yielding $dS = ||\mathbf{r}_s \times \mathbf{r}_t|| ds dt$. Thus, each infinitesimal step in the T domain corresponds to a differential area on the surface S . This substitution facilitates the evaluation of the surface integral via a double integral over a flat, rectangular region, which is typically more straightforward.

Change to Polar Coordinates

Transforming an integral into polar coordinates is advantageous when integrating over a circular region. To perform this substitution, let $x = r \cos \theta$ and $y = r \sin \theta$. The Jacobian determinant for this transformation is computed as follows:

$$\frac{\partial(x, y)}{\partial(r, \theta)} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & -r \sin \theta \\ \sin \theta & r \cos \theta \end{bmatrix} \quad (2.8)$$

The determinant is:

$$r \cos^2 \theta + r \sin^2 \theta = r \quad (2.9)$$

Therefore, the transformation becomes:

$$\int_S f(x, y) dS = \int \int_T f(x(r, \theta), y(r, \theta)) |r| dr d\theta \quad (2.10)$$

Given that the radius is always positive, Equation 2.10 simplifies to:

$$\int_S f(x, y) dS = \int \int_T f(x(r, \theta), y(r, \theta)) r dr d\theta \quad (2.11)$$

Change to Spherical Coordinates

To parametrise a spherical surface using spherical coordinates, the standard substitutions are applied: $x = \rho \sin \varphi \cos \theta$, $y = \rho \sin \varphi \sin \theta$, and $z = \rho \cos \varphi$.

$$\frac{\partial \vec{r}}{\partial \varphi} = \begin{bmatrix} \rho \cos \varphi \cos \theta \\ \rho \cos \varphi \sin \theta \\ -\rho \sin \varphi \end{bmatrix}, \frac{\partial \vec{r}}{\partial \theta} = \begin{bmatrix} -\rho \sin \varphi \sin \theta \\ \rho \sin \varphi \cos \theta \\ 0 \end{bmatrix} \quad (2.12)$$

The vector matrix is constructed to compute the cross product.

$$\begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ \rho \cos \varphi \cos \theta & \rho \cos \varphi \sin \theta & -\rho \sin \varphi \\ -\rho \sin \varphi \sin \theta & \rho \sin \varphi \cos \theta & 0 \end{bmatrix} \quad (2.13)$$

Computing the determinant yields:

$$\begin{bmatrix} \rho^2 \sin^2 \varphi \cos \theta \\ -\rho^2 \sin^2 \varphi \sin \theta \\ \rho^2 \sin \varphi \cos \varphi \end{bmatrix} \quad (2.14)$$

The magnitude of the resultant vector is:

$$\sqrt{(\rho^2 \sin^2 \varphi \cos \theta)^2 + (\rho^2 \sin^2 \varphi \sin \theta)^2 + (\rho^2 \sin \varphi \cos \varphi)^2} \quad (2.15)$$

which simplifies to:

$$\sqrt{\rho^4 \sin^2 \varphi \sin^2 \varphi + \rho^4 \sin^2 \varphi \cos^2 \varphi} \quad (2.16)$$

and further simplifies to:

$$\sqrt{\rho^4 \sin^2 \varphi} = \rho^2 \sin \varphi \quad (2.17)$$

Given that $\sin \varphi$ is non-negative for $0 \leq \varphi \leq \pi$, the absolute value is unnecessary for hemispheres. Therefore, the transformation yields:

$$\int_{\Omega} f d\omega = \int \int f \rho^2 \sin \varphi d\varphi d\theta \quad (2.18)$$

Consequently, for the unit sphere ($\rho = 1$), the formulation becomes:

$$\int_{\Omega} f d\omega = \int \int f \sin \varphi d\varphi d\theta \quad (2.19)$$

Relationship between Differential Solid Angle and Differential Area

Let the surface be parametrised by coordinates (ξ, η) . The position vector $\mathbf{r}(\xi, \eta)$ describes a point on the surface. The infinitesimal area element is:

$$dA = \left| \frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta} \right| d\xi d\eta \quad (2.20)$$

A point $\mathbf{r}(\xi, \eta)$ on the surface corresponds to a direction $\hat{\mathbf{u}} = \frac{\mathbf{r}}{|\mathbf{r}|} = \frac{\mathbf{r}}{r}$ on the unit sphere. The goal is to compute the differential solid angle $d\omega$, defined as the area on the unit sphere covered by small displacements $d\xi$ and $d\eta$. The partial derivatives of $\hat{\mathbf{u}}$ with respect to ξ and η are:

$$\frac{\partial \hat{\mathbf{u}}}{\partial \xi} = \frac{1}{r} \left(\frac{\partial \mathbf{r}}{\partial \xi} - (\hat{\mathbf{u}} \cdot \frac{\partial \mathbf{r}}{\partial \xi}) \hat{\mathbf{u}} \right) \quad (2.21)$$

$$\frac{\partial \hat{\mathbf{a}}}{\partial \eta} = \frac{1}{r} \left(\frac{\partial \mathbf{r}}{\partial \eta} - (\hat{\mathbf{a}} \cdot \frac{\partial \mathbf{r}}{\partial \eta}) \hat{\mathbf{a}} \right) \quad (2.22)$$

The differential solid angle is given by the magnitude of the cross product of the derivatives:

$$d\omega = \left| \frac{\partial \hat{\mathbf{a}}}{\partial \xi} \times \frac{\partial \hat{\mathbf{a}}}{\partial \eta} \right| d\xi d\eta \quad (2.23)$$

Substitute the derivatives:

$$\frac{\partial \hat{\mathbf{a}}}{\partial \xi} \times \frac{\partial \hat{\mathbf{a}}}{\partial \eta} = \frac{1}{r^2} \left[\left(\frac{\partial \mathbf{r}}{\partial \xi} - (\hat{\mathbf{a}} \cdot \frac{\partial \mathbf{r}}{\partial \xi}) \hat{\mathbf{a}} \right) \times \left(\frac{\partial \mathbf{r}}{\partial \eta} - (\hat{\mathbf{a}} \cdot \frac{\partial \mathbf{r}}{\partial \eta}) \hat{\mathbf{a}} \right) \right] \quad (2.24)$$

Let us denote the tangent vectors:

$$\mathbf{t}_\xi = \frac{\partial \mathbf{r}}{\partial \xi} - (\hat{\mathbf{a}} \cdot \frac{\partial \mathbf{r}}{\partial \xi}) \hat{\mathbf{a}}, \quad \mathbf{t}_\eta = \frac{\partial \mathbf{r}}{\partial \eta} - (\hat{\mathbf{a}} \cdot \frac{\partial \mathbf{r}}{\partial \eta}) \hat{\mathbf{a}} \quad (2.25)$$

These vectors $\mathbf{t}_\xi$ and $\mathbf{t}_\eta$ lie in the tangent plane of the unit sphere at $\hat{\mathbf{a}}$. Now compute:

$$\frac{\partial \hat{\mathbf{a}}}{\partial \xi} \times \frac{\partial \hat{\mathbf{a}}}{\partial \eta} = \frac{1}{r^2} \mathbf{t}_\xi \times \mathbf{t}_\eta \quad (2.26)$$

Since $\mathbf{t}_\xi$ and $\mathbf{t}_\eta$ are orthogonal to $\hat{\mathbf{a}}$, their cross product must be parallel to $\hat{\mathbf{a}}$:

$$\mathbf{t}_\xi \times \mathbf{t}_\eta = \alpha \hat{\mathbf{a}} \quad (2.27)$$

To find α , take the dot product with $\hat{\mathbf{a}}$:

$$\hat{\mathbf{a}} \cdot (\mathbf{t}_\xi \times \mathbf{t}_\eta) = \alpha (\hat{\mathbf{a}} \cdot \hat{\mathbf{a}}) = \alpha \quad (2.28)$$

Now expand using the definition of $\mathbf{t}_\xi$ and $\mathbf{t}_\eta$:

$$\mathbf{t}_\xi \times \mathbf{t}_\eta = \left[\frac{\partial \mathbf{r}}{\partial \xi} - (\hat{\mathbf{a}} \cdot \frac{\partial \mathbf{r}}{\partial \xi}) \hat{\mathbf{a}} \right] \times \left[\frac{\partial \mathbf{r}}{\partial \eta} - (\hat{\mathbf{a}} \cdot \frac{\partial \mathbf{r}}{\partial \eta}) \hat{\mathbf{a}} \right] \quad (2.29)$$

Now expand the cross product using the distributive property (and $\hat{\mathbf{a}} \times \hat{\mathbf{a}} = 0$):

$$\mathbf{t}_\xi \times \mathbf{t}_\eta = \left[\frac{\partial \mathbf{r}}{\partial \xi} - (\hat{\mathbf{a}} \cdot \frac{\partial \mathbf{r}}{\partial \xi}) \hat{\mathbf{a}} \right] \times \left[\frac{\partial \mathbf{r}}{\partial \eta} - (\hat{\mathbf{a}} \cdot \frac{\partial \mathbf{r}}{\partial \eta}) \hat{\mathbf{a}} \right] \quad (2.30)$$

$$= \frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta} - (\hat{\mathbf{a}} \cdot \frac{\partial \mathbf{r}}{\partial \eta}) \left(\frac{\partial \mathbf{r}}{\partial \xi} \times \hat{\mathbf{a}} \right) - (\hat{\mathbf{a}} \cdot \frac{\partial \mathbf{r}}{\partial \xi}) \left(\hat{\mathbf{a}} \times \frac{\partial \mathbf{r}}{\partial \eta} \right) \quad (2.31)$$

Taking the dot product with $\hat{\mathbf{a}}$ (terms vanish because (e.g., $\hat{\mathbf{a}} \cdot (\frac{\partial \mathbf{r}}{\partial \xi} \times \hat{\mathbf{a}}) = 0$):

$$\hat{\mathbf{a}} \cdot (\mathbf{t}_\xi \times \mathbf{t}_\eta) = \hat{\mathbf{a}} \cdot \left(\frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta} \right) \quad (2.32)$$

Thus:

$$|\mathbf{t}_\xi \times \mathbf{t}_\eta| = \left| \hat{\mathbf{u}} \cdot \left(\frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta} \right) \right| \quad (2.33)$$

Let $\mathbf{n} = \frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta}$ be the surface normal vector. Then:

$$|\mathbf{t}_\xi \times \mathbf{t}_\eta| = |\hat{\mathbf{u}} \cdot \mathbf{n}| = |\cos(\theta)| |\mathbf{n}| \quad (2.34)$$

where θ is the angle between the surface normal and the radial direction $\hat{\mathbf{u}}$. Finally, substitute back:

$$d\omega = \frac{1}{r^2} |\mathbf{t}_\xi \times \mathbf{t}_\eta| d\xi d\eta \quad (2.35)$$

$$d\omega = \frac{1}{r^2} |\cos(\theta)| \left| \frac{\partial \mathbf{r}}{\partial \xi} \times \frac{\partial \mathbf{r}}{\partial \eta} \right| d\xi d\eta \quad (2.36)$$

$$d\omega = \frac{|\cos(\theta)|}{r^2} dA \quad (2.37)$$

2.2 Radiometry

Radiometry is a field of physics concerned with the measurement of electromagnetic radiation. In rendering, we deal with the propagation of visible light, which constitutes a subset of the electromagnetic spectrum. To render an image, renderers must be able to measure the amount of energy that flows into the camera.

Radiant flux quantifies the rate of radiant energy flow and is also termed radiant power. Power represents the rate of energy transfer or conversion per unit time (measured in Joules per second), which is equivalent to Watts. For instance, a surface emitting heat without gaining energy from its surroundings is losing energy. The rate at which this surface loses energy is its power output. Radiant flux is fundamental in rendering as it underpins the definition of power for light sources. A light source has a power rating, much like the light bulbs in our homes. However, as renderers only need to measure visible radiation, the power attributed to a light source should only include radiation from the visible spectrum. Alternatively, renderers could compute the full spectrum and, using photometric methods, render the visible spectrum at the final stage.

Irradiance is a measure of the amount of radiant flux incident upon a surface and has units of Wm^{-2} . Radiant exitance, sometimes called radiosity, is the amount of radiant flux leaving a surface and shares the same units as irradiance. An example of irradiance is solar irradiance, a measurement often provided by weather stations, which can be useful for estimating the power generated by solar panels.

Intensity measures the amount of radiant flux flowing through a solid angle. In rendering, it is typically used to model point lights, as this type of light source does not

Quantity	Symbol	Unit	Differential Form
Energy	Q	Joule (Ws)	N.A.
Radiant Flux	Φ	Watt (W)	$\frac{dQ}{dt}$
Irradiance	E	Wm^{-2}	$\frac{d\Phi}{dA}$
Radiosity	M or B	Wm^{-2}	$\frac{d\Phi}{dA}$
Intensity	I	Wsr^{-1}	$\frac{d\Phi}{d\omega}$
Radiance	L	$Wm^{-2}sr^{-1}$	$\frac{d\Phi}{dAcos\theta d\omega}$ or $\frac{dE}{cos\theta d\omega}$ or $\frac{dB}{cos\theta d\omega}$

Table 2.1 Radiometric quantities

possess a surface area.

Radiance measures the amount of radiant flux transmitted or received by a surface perpendicular to a given direction. It is therefore a directional quantity with units of $Wm^{-2}sr^{-1}$. Radiance is the most important quantity in a renderer, as it accurately depicts how much energy flows in a particular direction. A key property of radiance is that it remains invariant along straight lines; that is, it does not attenuate with distance within a vacuum. Furthermore, in photometry, the counterpart to radiance is luminance, a quantity often associated with perceived brightness.

Table 2.1 presents several key radiometric quantities, detailing their symbols, units, and differential forms. Given a function for radiance, one can derive all of the other key radiometric quantities. For example, assuming radiance is solely a function of the azimuthal angle ϕ , such that $L(\phi) = (\pi - \phi)^2$, the irradiance E incident upon a surface can be found by integrating with respect to the solid angle ω . Therefore:

$$E = \int_0^{2\pi} \int_0^{\frac{\pi}{2}} (\pi - \phi)^2 \cos(\theta) \sin(\theta) d\theta d\phi \quad (2.38)$$

$$E = \frac{1}{2} \int_0^{2\pi} (\pi - \phi)^2 d\phi \quad (2.39)$$

$$E = \frac{\pi^3}{3} \approx 10.335 Wm^2 \quad (2.40)$$

Furthermore, assuming a small tile with dimensions of 10cm by 10cm, the total power incident upon it can be deduced as follows.

$$\Phi = \int_0^{0.1} \int_0^{0.1} E dx dy = \frac{\pi^3}{3} \frac{1}{10} \frac{1}{10} = \frac{\pi^3}{300} \approx 0.10335 W \quad (2.41)$$

2.3 Surface Interaction Models

When light interacts with a surface, it is either reflected or transmitted into the material. The nature of this surface interaction is described by specific models. This section will

explain some of these models and their properties.

A BRDF describes the proportion of light reflected from a surface, which depends on both the incoming and outgoing directions. Figure 2.5 illustrates the scattering behaviour of a BRDF. On the left, the energy from a single incoming ray is scattered into multiple directions, while on the right, the energy from multiple incoming rays is accumulated into a single outgoing ray. A BRDF is denoted by the function f_r and depends on two variables: an incoming direction and an outgoing direction. The properties of a well-defined BRDF are enumerated below:

- **Non-negativity:** The range of f_r must be non-negative ($f_r \geq 0$).
- **Dimensionality:** It is a function of two dimensions, or four when parameterised using spherical coordinates.
- **Reciprocity:** It must obey Helmholtz reciprocity, i.e., $f_r(\omega_i, \omega_r) = f_r(\omega_r, \omega_i)$.
- **Energy Conservation:** For all incoming directions ω_i , the integral over the hemisphere must satisfy $\forall \omega_i, \int_{\Omega} f_r(\omega_i, \omega_r) \cos(\theta_r) d\omega_r \leq 1$.

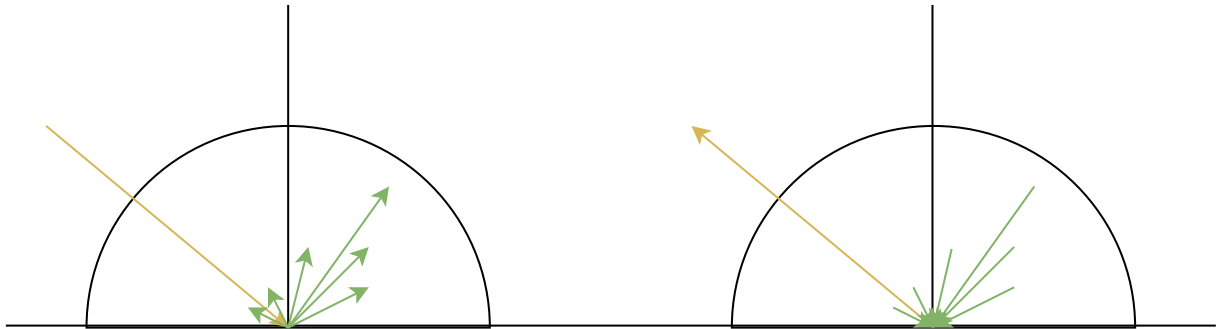


Figure 2.5 Left: An incoming yellow ray is scattered and distributed into multiple green rays. Right: Multiple incoming green rays contribute their energy to form a single outgoing yellow ray.

Bidirectional transmittance distribution functions (BTDFs) describe the proportion of light transmitted through a surface in a specific direction, given an incoming direction. BTDFs model phenomena such as refraction and consequently incorporate the Fresnel term. A bidirectional scattering distribution function (BSDF) can be conceptualised as a combination of the BRDF and BTDF. This implies that a BSDF can scatter light into any direction over the unit sphere, whereas the BRDF and BTDF are confined to scattering light over a unit hemisphere. More general forms, such as the bidirectional scattering-surface distribution function (BSSDF), also exist. In this model, light entering at one point on a surface may exit from a different point, thereby simulating sub-surface scattering.

2.3.1 BRDF Models

Montes and Ureña [6] classify BRDFs into three categories: empirical, theoretical, and experimental. Empirical BRDFs prioritise computational speed with no consideration for physical accuracy. Theoretical models are slower but simulate physics with high precision, while experimental ones are acquired using a goniophotometer or by approximating reflectance data with an analytical expression, such as through fitting. BRDFs are further classified as isotropic or anisotropic. Isotropic reflection occurs when light is reflected uniformly regardless of the material's rotational orientation. In contrast, anisotropic materials exhibit reflection that depends on this rotational orientation; brushed steel is a typical example. Figure 2.6 illustrates this distinction with two spheres, one featuring an isotropic material and the other an anisotropic one. Ideally, a BRDF should observe the following properties:

- **Physically Plausible:** It must be positive, reciprocal, and energy-conserving.
- **Expressive:** Its behaviour should be adjustable via intuitive parameters.
- **Usable:** It should be capable of representing a wide range of materials.
- **Realistic:** Its output should closely match nature, exhibiting effects such as directional diffuse reflection and Fresnel behaviour.

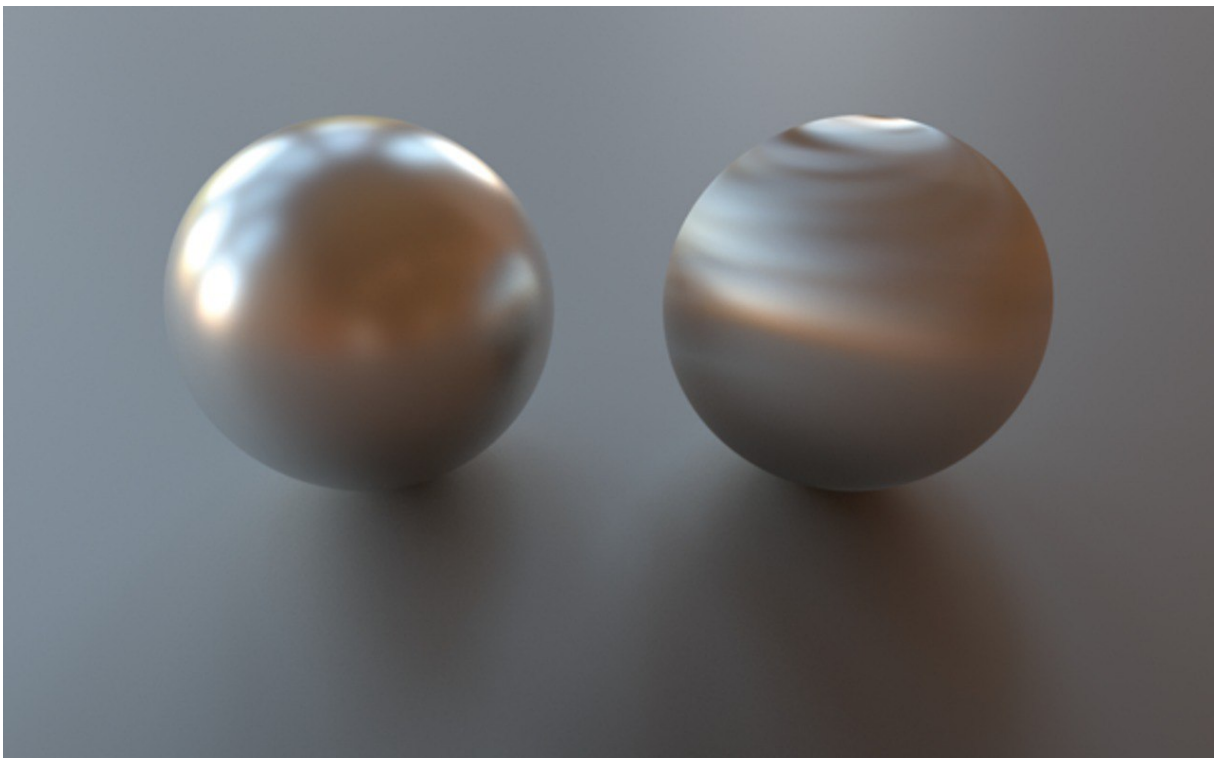


Figure 2.6 Left: Isotropic reflection; Right: Anisotropic reflection [7]

Lambertian BRDF

A diffuse material reflects light approximately uniformly in all directions. The simplest model for this behaviour is the Lambertian model, which assumes that incident light is scattered equally in every direction. It is defined as a constant function in Equation 2.42. The term ρ_d denotes the albedo, a value in the range $0 \leq \rho_d \leq 1$, where 0 signifies total absorption and 1 signifies total reflection. The factor of $\frac{1}{\pi}$ serves as a normalisation constant, ensuring energy conservation for this model.

$$f_r = \frac{\rho_d}{\pi} \quad (2.42)$$

The Lambertian BRDF is physically plausible; it is always positive, as it is a constant function. It automatically satisfies Helmholtz reciprocity and is energy-conserving. In the worst-case scenario, where $\rho_d = 1$, Equation 2.45 demonstrates that it is indeed energy-conserving.

$$\int_{\Omega} \frac{1}{\pi} \cos(\theta_r) d\omega_r \leq 1 \quad (2.43)$$

Applying Equation 2.19 yields:

$$\frac{1}{\pi} \int_0^{2\pi} \int_0^{\frac{\pi}{2}} \cos \varphi \sin \varphi d\varphi d\theta \leq 1 \quad (2.44)$$

$$\frac{1}{2\pi} \int_0^{2\pi} d\theta \leq 1 \implies 1 \leq 1 \quad (2.45)$$

Phong BRDF

The Phong BRDF [8] models a specular surface, which reflects light predominantly around the reflection vector. Real-world specular materials are not always ideal mirrors; they often exhibit perturbations that result in a glossy appearance. The Phong BRDF aims to model this behaviour. It is an empirical cosine-lobe model that remains in use in some applications today. The Phong equation is shown in Equation 2.46, where L is the viewer direction, R is the reflection vector (derived from the light direction), and n is the Phong exponent. A higher value of n results in a more mirror-like material. The constant k_s , in the range $0 \leq k_s \leq 1$, represents the ratio of specular reflectance to absorption.

The Phong BRDF is always positive, as it is clamped to zero when the angle between the reflection and viewer vectors exceeds 90 degrees. It obeys Helmholtz reciprocity but is not energy-conserving. The modified Phong model [9], given in Equation 2.47, introduces a normalisation constant, $\frac{n+2}{2\pi}$, which guarantees energy conservation. The modified Phong BRDF is therefore physically plausible.

Other Phong-based BRDFs exist, such as the Blinn-Phong model [10]. This variant uses the angle between the halfway vector and the surface normal instead of the reflection and viewing vectors (see Equation 2.48). This modification yields superior visual results, particularly when the angle between the reflected direction and the viewer exceeds 90 degrees. Equation 2.49 presents the Blinn-Phong BRDF with an added normalisation factor to ensure energy conservation¹.

$$f_r = k_s (R \cdot L)^n \quad (2.46)$$

$$f_r = \frac{k_s (n + 2) (R \cdot L)^n}{2\pi} \quad (2.47)$$

$$f_r = k_s (N \cdot H)^n \quad (2.48)$$

$$f_r = \frac{k_s (n + 2) (n + 4) (N \cdot H)^n}{8\pi (2^{-n/2} + n)} \quad (2.49)$$

Others

Other notable BRDF functions include the Cook-Torrance [11], Oren-Nayar [12], and GGX [13] models. The Cook-Torrance model was the first to introduce a class of models based on the concept of microfacets. In these models, rough surfaces are represented as a collection of microscopic surface elements, or microfacets. Surfaces that appear smooth are modelled with microfacets that are largely aligned, whereas surfaces that appear rough are modelled with randomly oriented microfacets. Furthermore, these microfacets can reflect light off one another and cast shadows on adjacent microfacets.

2.4 The Rendering Equation

The rendering equation [2] is a Fredholm integral equation of the second kind with no analytical solution. It describes the radiance L_o emitted from a surface point x in direction ω_o . To compute this value, the integral accounts for all incoming radiance L_i arriving at point x from every direction ω_i over the sphere centred at x . Equation 2.50 presents the more general form of the rendering equation, which also depends on time t and spectral frequency λ . The inclusion of λ allows for the modelling of spectral dispersion, while the time variable t enables the simulation of motion blur. The $\cos(\theta_i)$ term represents the cosine of the angle between the incoming radiance L_i and the surface normal at point x . This can also be expressed in dot product form as $\omega_i \cdot \hat{n}$. The domain

¹<http://www.farbrausch.de/~fg/stuff/phong.pdf>

of integration Ω typically represents either a hemisphere or a full sphere around point x . For example, when modelling reflection only, Ω is defined over the hemisphere. The term f_r is the BRDF, as detailed in Section 2.3.1.

$$L_o(x, \omega_o, \lambda, t) = L_e(x, \omega_o, \lambda, t) + \int_{\Omega} f_r(x, \omega_i, \omega_o, \lambda, t) L_i(x, \omega_i, \lambda, t) \cos(\theta_i) d\omega_i \quad (2.50)$$

More general forms of the rendering equation exist to model more complex phenomena, such as subsurface scattering. However, for our purposes, the formulation provided in Equation 2.50 is sufficient.

2.4.1 The Importance Equation

The adjoint quantity to radiance, known as *importance*, describes how sensitivity to light propagates backward from the sensor through the scene. It forms the basis of adjoint light transport algorithms such as light tracing and bidirectional path tracing. Unlike radiance, which quantifies the flow of light energy from emitters to sensors, importance expresses the relative sensitivity of the sensor to radiance arriving at a surface point from a given direction, indicating how strongly such radiance contributes to the final image.

The importance equation shares the same physical factors as the rendering equation i.e., visibility, geometry, and material reflectance, but evaluated in reverse. It is defined in Equation 2.51, where W is the importance function, W_e the emitted (or direct) importance (for a pinhole camera, typically 1 if visible, 0 otherwise), and f_r the BRDF which is used here under the assumption of Helmholtz reciprocity.

$$W(x, \omega_o, \lambda, t) = W_e(x, \omega_o, \lambda, t) + \int_{\Omega} f_r(x, \omega_i, \omega_o, \lambda, t) W(x, \omega_i, \lambda, t) \cos(\theta_i) d\omega_i \quad (2.51)$$

The total flux Φ measured by a sensor is then obtained by integrating the product of radiance and importance over all positions and directions, as shown in the measurement equation in Equation 2.52. This formulation shows that image formation results from the coupling of radiance (light transport) and importance (sensor sensitivity).

$$\Phi = \int_A \int_{\Omega} L(x, \omega) W(x, \omega) \cos(\theta) d\omega dA \quad (2.52)$$

2.5 Monte Carlo

Monte Carlo methods are employed to approximate the value of integrals that lack an analytical solution. The fundamental principle of these methods is that the expected value of the estimator converges to the true solution given a sufficient number of samples. Consider the integral I , where Ω is the domain of integration:

$$I = \int_{\Omega} f(x) dx \quad (2.53)$$

An estimate for I is obtained by averaging a number of samples, x_i , each weighted by the reciprocal of its probability density.

$$\langle I \rangle = \frac{1}{N} \sum_{i=1}^N \frac{f(x_i)}{p(x_i)} \quad (2.54)$$

From probability theory, the expected value of the estimator $\langle I \rangle$ is

$$E[\langle I \rangle] = E\left[\frac{1}{N} \sum_{i=1}^N \frac{f(x_i)}{p(x_i)}\right] \quad (2.55)$$

$$E[\langle I \rangle] = \frac{1}{N} \sum_{i=1}^N E\left[\frac{f(x_i)}{p(x_i)}\right] \quad (2.56)$$

$$E[\langle I \rangle] = \frac{1}{N} N \int_{\Omega} \frac{f(x)}{p(x)} p(x) dx \quad (2.57)$$

$$E[\langle I \rangle] = \int_{\Omega} f(x) dx \quad (2.58)$$

$$E[\langle I \rangle] = I \quad (2.59)$$

2.6 Path Tracing

Path tracing is a Monte Carlo rendering technique that simulates global illumination by statistically estimating the light transport equation. At its core, it approximates the integral of radiance arriving at a camera by randomly sampling light paths through the scene. For each pixel, multiple rays are traced from the camera into the scene, bouncing off surfaces according to material properties (e.g., diffuse, specular) and terminating probabilistically via Russian roulette. Each path's contribution is weighted by its probability, ensuring an unbiased estimate of the pixel's colour. Since light transport is inherently high-dimensional, brute-force computation is infeasible. Monte Carlo integration provides a practical solution by using random sampling to converge toward the correct solution over time.

2.7 Candela

Candela [14] is a CPU-based, cross-platform rendering engine implemented in C++. Designed for clarity and extensibility, it supports loading geometry and lighting configurations while providing a straightforward codebase for modification. The renderer includes advanced features such as multithreaded path tracing, tiled rendering, and variance-based adaptive sampling, yet maintains a modular architecture that simplifies the integration of new rendering techniques. Its current shading model focuses on fundamental material types (e.g., diffuse and specular), avoiding unnecessary complexity. This combination of simplicity and adaptability makes Candela particularly suitable for research-oriented rendering projects, where a clean foundation is essential for prototyping and evaluating novel algorithms.

2.8 Software Design

The ECS is a data-driven design pattern popular in game architecture [15] and other performance-critical software systems, such as simulations. Scott Bilas [16] introduced the idea of a data-driven approach to game design in a presentation where the problems resulting from the use of object-oriented programming (OOP) in the context of Massively Multiplayer Online (MMO) games were discussed. Bilas explained that OOP often leads to a deep inheritance hierarchy that inhibits code modification. To circumvent this, developers sometimes resort to shortcuts such as hoisting, which results in large, monolithic classes. In such games, the use of OOP can lead to a database being hard-coded into a class hierarchy. Instead, Bilas introduced the concept of a *component system*, where each component is a self-contained piece of game logic. The *game object*, an item that represents a particular object or element in a game, stores a list of such components. The key idea is that an application whose functionality is centred around data cannot work by data-driving object properties but by data-driving the structure of objects.

Martin [17] furthered Bilas's work by refining the component system into an ECS. Martin defines an *entity* to simply be a globally unique identifier (GUID), a *component* to store raw data that provides an aspect to an entity and finally a *system* where all logic acting on component data is placed. ECS is powerful when data is the main driver, because it allows one to change an object's capabilities simply by adding and removing components. Achieving the same flexibility in classic OOP would necessitate modifications to the class hierarchy. ECS offers improved performance through cache-friendly data organisation, enhanced flexibility, and better code reusability. Its principles align with data-oriented design, emphasising efficient data layout and pro-

cessing. This makes ECS well suited for scenarios involving large numbers of objects or entities and has expanded to other domains requiring high performance and scalability, as seen in its adoption by modern game engines and software frameworks.

2.8.1 Software Quality Attributes

Software quality attributes facilitate the evaluation of a system's performance with respect to specific characteristics. They also enable the comparison of different systems and are instrumental in guiding design decisions.

Extensibility is a quality attribute that measures the ease with which software can be expanded over time. Szyperski and Clemens [18] state that extensible systems are modular, and that independent extensibility is a desirable property. This means that extending an application does not require global knowledge, and that extensions are mutually exclusive, i.e., they do not depend on one another. One method for quantifying extensibility in an application is to count the number of extensible methods.

Flexibility is another key software quality attribute. Software is considered flexible if it can be readily modified to incorporate new functionality for use in different domains. Flexibility is directly related to the volume of code that is affected by a change. Eden et al. [19] quantify flexibility by introducing the concept of evolution complexity. An evolution function is used to characterise development steps, which are then evaluated using a cost metric. They conclude that flexibility is indeed related to the magnitude of change within the code base, and that while an implementation may be flexible with respect to one class of changes, it is inherently inflexible towards others.

2.9 WebSockets

WebSocket is a communication protocol enabling full-duplex, bidirectional communication between clients and servers over a single TCP connection. Conceived by Ian Hickson and Michael Carter in 2008 as part of the HTML5 initiative [20], it was standardised by the Internet Engineering Task Force (IETF) in 2011 [21]. WebSocket addressed limitations of HTTP-based solutions for real-time web applications, evolving from earlier techniques like Comet [22] and long polling. The protocol has gained widespread adoption across major web browsers and server technologies, becoming fundamental for building interactive and real-time web applications such as live chat, gaming and financial trading platforms [23].

2.10 Spectral Transforms

The 2D Discrete Fourier Transform (DFT) is a mathematical operation that converts a 2D function or image from the spatial domain to the frequency domain. In Fourier transforms, the signal is decomposed using sine and cosine functions as fundamental components. The frequency domain allows for filtering and subsequent transformation back to the spatial domain using the Inverse Fourier Transform. Common filters include low-pass and high-pass filters. When filtering an image for denoising purposes, certain frequencies, potentially associated with noise, are reduced to produce a clearer image. However, Fourier transforms have a global nature, meaning that changing a single pixel in one domain can impact the entire image when reverting to the other domain. For example, applying a low-pass filter to the frequency domain of an image results in the attenuation of that frequency throughout the spatial domain of the image.

Discrete Wavelet Transforms (DWTs) [24, 25] enable signal analysis across multiple resolutions. By employing the wavelet transform, it is possible to scrutinise the frequency characteristics of an image at specific locations. Furthermore, precise frequency filtering can be executed, allowing for the selective filtration of frequency bands within particular regions of the image. At a fundamental level, wavelets function through decomposing the image using both a low-pass filter (representing average coefficients) and a high-pass filter (representing detail coefficients). The low-frequency components are iteratively subdivided until the desired frequency resolution is achieved. There are a variety of wavelets to select from, with the choice being contingent upon the intended application and desired attributes. Essentially, different wavelets excel at extracting distinct signal features; for instance, one may be more adept at isolating noise, while another may be more effective at image compression.

Curvelets [26] extend the functionality of wavelets with a focus on maintaining curve-like structures within images. Unlike wavelets, which decompose images along horizontal, vertical, and diagonal axes, curvelets can analyse images at diverse angles. Curvelets introduce the notion of wedges, which represent angular sectors in the frequency domain. These wedges enable directional filtering of images, similar to spatial filtering in wavelets. Consequently, curvelets are well-suited for addressing rendering phenomena that prominently feature curves, such as caustics.

2.11 Search

Bayesian optimisation with Gaussian processes represents a hyperparameter search technique [27]. This method constructs a model of the function's appearance by establishing a confidence interval along the mean of the function between sampled points.

It proves particularly beneficial when dealing with costly functions to minimise, as it reduces the number of necessary calls.

The Nelder-Mead [28] algorithm represents a direct search approach (without utilising gradients). It employs a simplex, which changes shape throughout iterations until all vertices converge, facilitating minimisation. While straightforward, it may exhibit slow convergence, particularly in higher dimensions. However, it demonstrates robustness to noise and is well-suited for non-smooth functions. Powell's [29] method, on the other hand, operates as a direct search conjugate direction technique. It navigates towards a minimum by traversing computed directions across the domain. This method proves efficient for smooth functions and higher dimensions. COBYLA [30] constitutes another direct search method, employing linear approximations of the objective function. It can also be applied to constrained minimisation tasks. However, its linearity may result in lower accuracy compared to alternative methods.

Non-direct search methods such as limited-memory Broyden-Fletcher-Goldfarb-Shanno with boundaries (L-BFGS-B)[31, 32], truncated Newton conjugate gradient (TNC)[33], sequential least squares programming (SLSQP)[34], and trust-constr[35, 36] typically approximate the gradient to facilitate the search. L-BFGS-B, a variant of Broyden-Fletcher-Goldfarb-Shanno (BFGS) with limited memory and bounded constraints, employs a quasi-Newton approach to approximate the Hessian, a matrix of second-order partial derivatives describing the function's curvature at a point. It is well-suited for smooth, convex functions and performs effectively in high-dimensional scenarios. SLSQP, also a quasi-Newton method, excels when the objective function is smooth and differentiable but may not scale as efficiently as L-BFGS-B in higher dimensions. TNC is adept at handling large-scale problems by utilising the conjugate-gradient method to approximate the gradient, demonstrating better scalability to higher dimensions and faster convergence for well-conditioned problems. trust-constr utilises a trust-region approach, employing a model of the function (typically linear or quadratic) considered sufficiently accurate. The region expands if the model is deemed reliable and contracts otherwise, making this algorithm more robust and suitable for ill-conditioned functions.

2.12 Image Quality Metrics

Image quality metrics assess different visual aspects of an image. Mean Squared Error (MSE) yields a high score when the content of two images is similar but their structure differs significantly. However, it is prone to outliers because larger differences are penalised more due to the quadratic nature of the formula. Peak signal-to-noise ratio (PSNR) calculates the peak signal-to-noise ratio, with a higher MSE resulting in a

better PSNR score, and vice versa. Structural Similarity Index (SSIM) is a perception-based model that considers three distinct similarities: luminance, contrast, and structure. These components are relatively independent and are amalgamated to form the structural similarity index [37]. Unlike MSE, SSIM is less affected by differences in brightness, as this only influences the luminance component of this metric.

Modern approaches may be more suitable for images featuring high dynamic range (HDR), which is typically produced by renderers. High-Dynamic-Range Visual-Difference-Predictor v3 (HDRVDP3) [38] stands out as a modern image comparison metric that integrates a precise model of the human visual system. It simulates how the eye responds to diverse stimuli, including contrast sensitivity, luminance masking, luminance adaptation, and even considers pupil ageing. The key aspect of this metric lies in its neural model for contrast sensitivity and contrast masking, which predicts the visual system’s ability to detect and distinguish patterns. Unlike its predecessors, HDRVDP3 introduces an additional parameter known as the task, which, in our context, is configured as side-by-side. As implied by the task name, this setting evaluates images as if a human were comparing them directly alongside each other.

FLIP [39, 40] is specifically tailored for images generated by rendering software. It assesses the perceptual difference experienced by humans when viewing two rapidly alternating images. The rationale behind this method stems from the common practice among graphics researchers who analyse and evaluate image quality in this manner. According to the authors, FLIP is adept at detecting differences in point-like structures such as fireflies, which holds significance in Monte Carlo-based rendering techniques. Originally designed for low dynamic range (LDR) images, HDR-FLIP overcomes this limitation by generating multiple LDR images from an HDR image through exposure compensation and tone-mapping. These LDR images are then merged to derive a final score.

Building on these perceptual models, the Learned Perceptual Image Patch Similarity (LPIPS) metric [41] takes a data-driven approach by leveraging the internal activations of deep convolutional neural networks. The key finding is that the feature representations within a network, even across different architectures (SqueezeNet, AlexNet, VGG) and levels of supervision (supervised, self-supervised, or unsupervised), provide an embedding where distance correlates remarkably well with human perceptual judgements. This suggests that perceptual similarity is an emergent property of deep visual representations tuned to predict structure in the world. The LPIPS metric computes the distance between two images in this deep feature space, and its performance can be further improved by “calibrating” the features on a large-scale dataset of human judgements (the BAPPS dataset)]. It was shown to outperform previous metrics like SSIM by a large margin on a variety of distortions and real algorithm outputs.

2.13 Denoising

Spatiotemporal Variance-Guided Filtering (SVGF) [42] is a denoising technique applied over time, interpolating reprojected values between consecutive frames through a combination of bilinear filtering and exponential moving average. While temporal luminance variance is calculated over time, spatial estimation is employed for disocclusion cases, utilising the G-buffer to steer a bilateral filter. This process further engages a hierarchical à-trous wavelet filter[43] to appropriately denoise or blur the luminance. The luminance is then modulated back to generate the final image.

ReLAX [44], developed by NVIDIA, serves as an enhancement to SVGF. While initially tailored for compatibility with the ReSTIR algorithm, it is not restricted to it. ReLAX refines various stages of the SVGF pipeline and introduces novel elements. It partitions the signal into diffuse and specular components due to their distinct behaviours. For example, handling motion vectors differs since specular reflections are view dependent. Despite this differentiation, denoising for both components is concurrently computed. Surface modulation for each component also undergoes distinct computations, for instance, for the specular component a pre-integrated specular BRDF is required. Furthermore, temporal reprojection employs a different approach, using plane distance instead of depth difference to validate reprojection. Bicubic resampling is now integrated during reprojection for certain scenarios. Notably, large disoccluded areas are addressed with a sparse filter instead of the bilateral filter, broadening the regions where approximation samples are gathered. To mitigate responsiveness issues related to history accumulation, ReLAX implements fast history accumulation, accumulating data from 3 to 10 frames and employing a method inspired by Temporal Anti-Aliasing to clamp normal history (which may extend up to 50 frames) to fast history. Moreover, ReLAX incorporates a rank-conditioned rank-selection filter for firefly suppression. Additionally, unlike SVGF, additional edge-stopping functions are utilised for the specular component.

ReBLUR [45] employs a hierarchical denoising approach based on recurrent spatiotemporal filtering, where the denoised outcome from the preceding frame acts as the historical reference in the current frame. This strategy reduces the computational expense of spatial filtering per frame. Furthermore, ReBLUR is compatible with checkerboard rendering. The workflow of ReBLUR begins with a pre-blur stage, designed to soften outliers. Subsequently, temporal accumulation occurs, allowing for the accumulation of data from up to 64 frames. ReBLUR then generates the initial four MIP levels of the accumulated diffuse, specular, and viewZ buffers, which are utilised in the history fix pass. In instances where historical data is absent, such as disocclusion scenarios, instead of employing a broad spatial filter, which is computationally demanding, diverse MIP levels are employed for history reconstruction. Following this, ReBLUR applies a

dual-pass blur to the output and conducts temporal stabilisation, which also includes anti-lag mechanisms as it maintains its own historical data.

NVIDIA's OptiX AI-accelerated denoiser [46] utilises deep convolutional networks to reconstruct Monte Carlo global illumination images from highly noisy inputs, specifically tailored for scenarios with limited samples per pixel. The neural network undergoes training with tens of thousands of images per scene, with recurrent neural networks employed to ensure temporal stability and to gather additional illumination details over time. Within the autoencoder architecture, each layer comprises recurrent blocks that decrease resolution capacity by a quarter through subsampling during encoding, and then upsample by 4 during decoding. Each recurrent block is composed of three convolution layers: the first receives input from the preceding block and the first in the whole chain receives input from the noisy RGB, normal, depth, and roughness textures from the current frame. The second and third layers additionally take input from the hidden block in the encoding layer, representing the state of the recurrent block from the previous frame, thus contributing to temporal stability by providing feedback. Skip connections are integrated to link encoder and decoder layers, aiding in the retrieval of lost information during the encoding and decoding processes.

2.14 Summary

This chapter provided a comprehensive overview of the fundamental concepts underpinning modern rendering. We began with the mathematical and radiometric principles necessary to quantify light transport, then explored the physical models governing surface interactions and the formalisms describing global illumination. The discussion extended to the statistical methods, like Monte Carlo integration and path tracing, used to solve these problems computationally. Finally, we surveyed the software design patterns, image quality metrics, and advanced denoising techniques that are essential for building efficient, high-fidelity rendering systems. Together, this background establishes the necessary foundation for the contributions presented in the subsequent chapters of this thesis.

3 Literature Review

This chapter surveys the field of light transport simulation, with a critical focus on the persistent challenge of rendering caustics in real-time. The objective of this review is to identify and justify the selection of an algorithmic approach that is suitable for high-performance implementation on modern GPU architectures.

The analysis begins by examining the foundational landscape of unbiased Monte Carlo methods, from path and light tracing to the more robust bidirectional path tracing. This establishes the core sampling problem that all subsequent methods aim to address. We then analyse biased but consistent techniques, such as photon mapping and radiosity, which trade theoretical perfection for practical efficiency.

The discussion then progresses to modern offline solutions, particularly specular manifold exploration, which represents the current state-of-the-art for solving complex caustic paths. Furthermore, we investigate contemporary real-time frameworks like RESTIR, acknowledging their transformative impact on general global illumination while critically assessing their specific limitations in handling high-frequency caustic paths.

Finally, the chapter concludes by reviewing supporting work in adjacent fields that enable our contributions: tools for debugging and profiling rendering pipelines, specialised techniques for real-time caustics, evaluation frameworks, and denoising methods tailored for caustic signals.

3.1 Ray-based

The use of ray-based techniques in the context of computer graphics first appeared in 1968 [47] where ray casting was used both to determine whether points in world space are occluded and to reproject a surface point to the eye. The time spent to render an image was about 30 minutes and Appel [47] improved this by casting rays on the contour lines of the occluders, in order to project shadows more efficiently and then projected the scene in the image plane line by line using the cutting-planes method. Interestingly, the term ray casting was coined by Roth [48].

Ray based reflections and refractions were first explored by Whitted [49], where the ray casting technique was repeatedly used at each intersection to gather light and is today known as recursive ray tracing. Whitted paved the way for further techniques such as distributed ray tracing where multiple rays are used to render phenomena such as depth of field and soft shadows [11]. This showed how powerful ray based algorithms were, because not only could they be used for simple visibility tests, but also for light transport.

In 1986, the light transport problem was formalised by Kajiya [2] with the ren-

dering equation. The rendering equation describes a recursive integral which mirrors Whitted's recursive ray tracing and leads to path tracing. In addition to this, Kajiya discussed several methods to approximate this equation, including but not limited to Monte Carlo integration.

Path tracing is probably the most naive way of approximating the rendering equation. A ray starts from the camera and is traced until an object is hit in the scene. The surface material at the hit point is queried and it is first determined whether the surface is emissive. If it is, the radiance is added to the result. Then, from the point on the hit surface, a new ray is traced into the scene and the process is repeated until a condition is met. If the repetition is stopped abruptly by simply limiting the depth, the result will be biased. Consider the case where rays are terminated at depth n . This means that all paths of depth $n + 1$ and over are culled and the result of such rays is assumed to be zero. In other words, this means one would simply be computing the first $n + 1$ terms of the Neumann representation of the rendering equation. In order to keep the result unbiased, Russian roulette can be used to terminate the repetitive process. Russian roulette works by assigning a non-zero probability that decides whether a new ray will be cast. Generally, the more the ray depth increases, the more likely it is to terminate a path. This way, in theory, paths of any length can be sampled. This method works because the result is divided by the probability of taking a ray. Therefore, a rare path will be weighted more than a common path.

Although path tracing generates the correct solution, convergence is slow even for offline use. A variation of this algorithm uses NEE where at every surface hit point, shadow rays are launched towards emissive surfaces. Radiance calculation is made possible by using the surface formulation of the rendering equation where the integral is taken with respect to all surfaces in the scene. In turn, this is made possible using the substitution $d\omega = \frac{\cos\theta dA}{r^2}$. This variation drastically improves convergence for scenes with a large number of diffuse surfaces. However, due to the r^2 term in the denominator, the variance of the result increases due to the approaching singularity when r is close to zero [50].

Light tracing was introduced by Arvo et al. in 1986, at the time known as backward ray tracing [51]. This method uses ray tracing but, instead of starting from the camera, rays are traced starting from the light sources. An illumination map is used as a pre-pass to store the amount of energy that is deposited on a partially diffuse surface. Whenever energy arrives on a particular texel (texture pixel), the energy is bilinearly distributed among the four corner points (data points) making up the surface. Querying the intensity on a point of the surface is done in a similar fashion by interpolating the four data points. The intensity at each data point is then calculated by dividing the accumulated energy by the unit surface area. The illumination map is further populated by repeating this process recursively. After it is constructed, a renderer can simply add

its values to the diffuse component when querying the related surface.

Light tracing has the advantage of rendering caustics well in certain scenes. However, there are scenes where both path tracing and light tracing cannot render difficult paths efficiently. Figure 3.1 shows such a scene where a light source is enclosed inside an open box. The open side of the box faces the ceiling and there is only a small gap where light can escape. In the case of path tracing, there will be very few paths that make it inside the box. For light tracing, this scenario is a bit easier; some light rays will manage to escape however convergence will still be low.

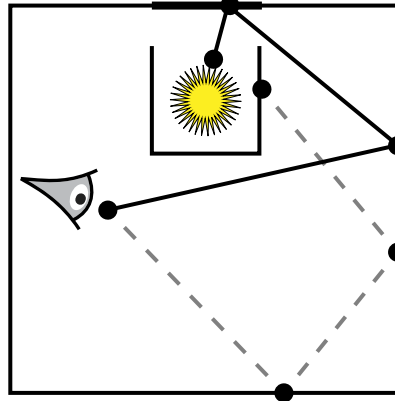


Figure 3.1 Difficult case for path tracing (adapted from [7])

Lafortune & Willems [9] tackle this issue with bidirectional path tracing, a technique that combines path and light tracing together. The algorithm works by firing a ray from the camera and letting it propagate through the scene until the path is terminated using russian roulette, as is normally performed with naive path tracing. All the intersection points are saved in a list. Then, a ray is fired starting from a light source and is also let to propagate in the scene. Again, the intersection points are saved in a list. The intersection points from each list are then joined together in all possible ways using shadow rays and a weight is associated with each connection. Then, all the connected paths's contributions are added to the image.

Figure 3.2 illustrates the algorithm. The left path (light, x_0 to x_3) is connected with the right path (eye, y_0 and y_1) using multiple shadow rays. The weights (C_{xy}) are used so that when multiple paths of the same length are produced, the energy is not duplicated. For example, path y_0, x_2, x_1, x_0 and path y_0, y_1, x_1, x_0 have length 4. Their weights, C_{21} and C_{12} must add up to 1. If no weights are applied, it would be as if we generated two samples for paths of length 4, and we would be doubling the energy. In general, as long as equation 3.1 holds, any weights can be selected.

$$\forall N : \mathbb{N}, \sum_{i=0}^N C_{i,N-i} = 1 \quad (3.1)$$

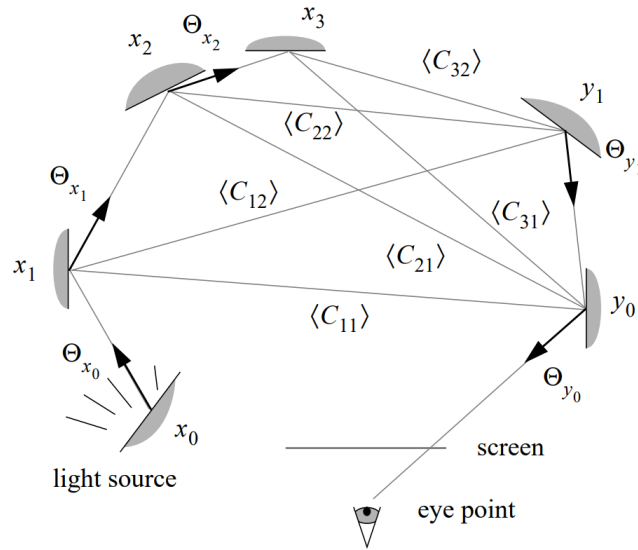


Figure 3.2 Bidirectional path tracing (adapted from [9])

The liberty to select weights implies that multiple sampling strategies can be used. A strategy may be weighted more than another one, depending on materials or other events. Modern bidirectional path tracers make use of multiple importance sampling, where more than one sampling technique is used to evaluate a given integral [52].

3.2 Radiosity

The radiosity algorithm introduced by Goral et al. [53] is a method that performs shading in scenes consisting mainly of diffuse materials. The surfaces describing the scene are subdivided into small patches. The greater the number of patches, the more accurate the result. For each patch, the method calculates the amount of radiosity that the other patches are contributing to the current patch. This is achieved by calculating the form factor, the fraction of energy leaving one patch and arriving at the other one. While this method produces realistic results, it is biased. This bias stems from the fact that the scene is discretised and is inversely proportional to the number of patches. However, this method is not stochastic and therefore it is not prone to the grainy noise that is usually observed in stochastic methods. A better global illumination solution in this case requires the addition of other methods such as path tracing to compute view-dependant effects like specular reflections.

3.3 Photon Mapping

Photon mapping [54] is a two-pass method that is similar to light tracing. In the first pass, the photon map is constructed. A predetermined number of photons is fired from all the light sources into the scene and are stored in a photon map, which is a data structure that stores the position that the photon hit, the energy deposited, and the direction that the photon came from. Jensen et al. recommend a kd-tree as the data structure for the photon map. The directional distribution of the emitted photons is proportional to the power/direction curve of the emitter [55]. For instance, for area light sources, power is cosine distributed with respect to the polar angle (Lambert's cosine law). For efficiency, Russian roulette is used to determine whether a photon is absorbed, reflected or transmitted.

The second pass is performed during shading. Using the photon map, the shader queries the nearest photons around the point of interest, which is chosen by tracing a ray from the eye into the camera. The direction and power together with the surface BRDF are then used to estimate the radiance at that point within the surrounding sphere. One problem with this method is that energy may leak from positions that are beyond reach. Figure 3.3 illustrates the problem. Photons on the other side of the wall are being included in the radiance estimation step. To improve this estimate, a filter kernel can be used. For instance, a kernel could weigh photons near the interest point higher than farther ones. Kang et al. [56] summarise a number of kernel filters in their state-of-the-art paper. These include Cone, Epanechnikov, Quartic and Gaussian filters. As the rendering equation can be split into several components, so can the photon maps. This gives the advantage of concentrating more photons where is needed the most. Typically, caustics maps are more dense than indirect lighting maps, and specular reflections are still ray based.

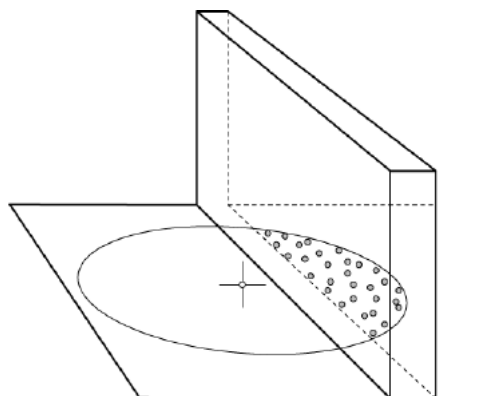


Figure 3.3 Energy leak in photon mapping (adapted from [57])

The photon mapping algorithm is biased but consistent. Increasing the number of photons reduces the bias and theoretically an infinite number of photons would make

the algorithm unbiased. Additionally, the lookup region affects both the variance and bias. If the radius of the sphere is decreased, bias decreases and variance increases while if the radius is increased, bias increases and variance decreases.

There have been numerous advancements and variations in photon mapping. These include photon splatting [58], photon tracing [59], progressive photon mapping [60] and photon relaxation [61] [56]. In photon splatting, it was observed that part of the total photon contribution could be precomputed using a texture attributed with a surface. Photons are splatted and the splat disc size represents the contribution of the photon (power) which spans multiple pixels. Triangles enclosing the splats are added on the plane of the hit surface and are mapped to the texture. During rendering, the triangle's colour is set to the power of the photon multiplied by the BRDF and the splat is added to the frame buffer using blending. This method fits nicely in the GPU pipeline. Photon tracing methods are focused on the emission techniques, for example selectively firing photons in regions of interest. Progressive photon mapping consists of an initial ray tracing pass to find which surfaces are visible through the camera. Then the photons are traced into the scene from the light sources, and the photons that are within the region of a pixel are accumulated. This photon tracing pass is repeated until the scene converges. It is important to note that in progressive photon mapping, the photons need not be stored, and therefore there are no space limitations. In photon relaxation the goal is to reduce noise without increasing bias. While usually this is addressed at the kernel level, photon relaxation directly manipulates the photons stored in the map. It works by repelling photons from each other to evenly distribute them, using feature detection to be able to keep high frequency details. The result is a distribution with a blue noise spectral signal structure which greatly reduces noise.

3.4 Manifold-based

Specular Manifold Sampling (SMS) is a unified manifold sampling strategy for rendering reflective and refractive caustics. This technique permits unbiased search on a specular surface using a Newton solver given the path endpoints and the surface constraints. However, the Newton solver in previous manifold next event estimation (MNEE) methods converges to one of the potentially many solutions, the closest one to the solution point. Every solution point has a basin which is a subset of the surface manifold. When the Newton solver's initial point is contained in this basin then it converges to that one point. Zeltner et al. [62] propose a solution to explore the whole space without introducing bias. Doing this requires knowing the probability of randomly selecting the basin. Estimating this is not straight forward since the probability estimate is in the denominator but this problem is tackled using [63], where the de-

nominator is transformed into a geometric series, essentially moving the calculation of the estimation of this probability value to the numerator. Zeltner et al. provide a biased but consistent version of this where the number of trials are limited, resulting in energy loss. Additionally, the surface constraints were improved. The original constraints proposed by Jakob and Marschner [64] converge to points that are back-facing and thus discarded (because the constraints do not discriminate between front and back-facing). Zeltner et al. changed the constraints to measure the difference between the incoming and outgoing ray directions instead of using the half vector.

Xu et al. [65] introduce a novel method for rendering caustics. They exploit the observation that specular paths (and basins) tend to be similar among the neighbouring shading points, and can be used as a seed to initialise the specular manifold walk which significantly reduces the time take to process the Newton solver. The reuse can also be done temporally. This technique is based on the biased version introduced by Zeltner et al. [62], and therefore it is also biased (and consistent). The results show that this method is better than the biased SMS. However, this method is effective with small light sources as larger ones reduce the similarity reuse and also small roughness values. Additionally, larger areas of reuse do not improve image quality proportionally.

SMS samples specular paths depending on the size of the convergence basins and ignores their energy distributions (critical for various reduction). Fan et al. [66] introduce manifold path guiding where sub-paths comprised of a specular chain and two non-specular separators are studied. Their method learns specular manifolds by temporally reusing sub-paths, and performing manifold walks to obtain new sub-paths. It exploits the coherence among sub-paths formed by specular manifolds, enabling the reconstruction of continuous energy distributions for better sampling. Their method outperforms previous unbiased rendering methods by reducing variance by more than one order of magnitude (40 times less). Connecting chains is a challenge and one major concern in this paper. This method only handles cases where one end of the chain lies on the light.

Granizo et al. [67] present a new algorithm for interactive rendering of caustics. The authors introduce a dimension reduction technique that simplifies the search for light paths to a one-dimensional curve (using coplanarity) on the surface, making it possible to render caustics interactively. This approach not only speeds up the rendering process but also simplifies implementation by eliminating the need to compute path derivatives. The paper demonstrates the algorithm's effectiveness in producing high-quality caustics quickly, significantly outperforming existing methods like Photon Mapping and Practical Path Guiding, especially in complex scenes with high tessellation levels. The algorithm maintains temporal consistency, producing noise-free and flicker-free animations of underwater caustics. However, it is limited to purely specular caustics and point light sources, and may not work as well with glossy surfaces or

area light sources. However only one solution (basin) is explored, similar to MNEE, and therefore accuracy is traded for speed and noise reduction. This means this solution is biased.

3.5 Resampling

Reservoir sampling [68] is an algorithm used to sample a subset (reservoir) of k elements using a stream of samples selected from a probability distribution. Every witnessed element can end up in the reservoir with the probability distribution of what has been witnessed so far. In Monte Carlo rendering the weighted reservoir sampling with replacement variant is used, which means that an element may be witnessed again. If all the possible elements are witnessed, the probability distribution of the buffer (the subset) will be approximately proportional (or equal if no repetition occurs) to the original probability distribution. This means that the buffer will effectively become a subset of k randomly (weighted) picked elements.

Importance resampling (resampled) [69] is a technique used to sample a distribution defined by a PDF, similar to cumulative density function (CDF) inversion and rejection sampling it is used for importance sampling. However, importance resampling generates samples that are approximately distributed according to the desired PDF. It works by selecting a number of M samples, which are stored in a list. These samples are then resampled with a particular probability which makes the resulting samples approximately distributed with the desired PDF. When used alone the discrete samples introduce bias, but when used in conjunction with importance sampling it serves to reduce variance (under several assumptions which are generally true).

ReSTIR [70] introduce a method to sample one-bounce direct lighting from many lights, suitable for real-time ray tracing with fully dynamic scenes. Reservoir sampling in conjunction with resampled importance sampling are utilised to make use of distributions which have already been sampled either in space (neighbouring pixels) or in time (neighbouring pixels after temporal reprojection using motion vectors). Reservoir sampling is used to feed resampled importance sampling (RIS), which greatly reduces memory constraints. Previously, lots of samples needed to be retained. The authors propose two methods; one unbiased and the other biased, and show that unbiased and biased offer a speed-up of 6 to 60 and 35 to 65 respectively on selected scenes (Amusement Park, Subway, Bistro and Zero Day).

Ouyang et al. [71] extend ReSTIR to ReSTIR global illumination (GI), allowing global illumination. In contrast to ReSTIR that samples from the lights, ReSTIR GI generate samples by tracing rays from visible points (points that are visible from the camera) and uses the hit-point as an initial sample. Then path tracing is used so that path

samples are reused. Since ReSTIR GI sampling scheme is based on the visible point's position (and not just the light source), the solid angle's PDF must be weighted accordingly using the Jacobian determinant [72] of the transformation for spatial re-use. Not needed for temporal re-use as the source and target pixels are supposed to be the same. Additionally, resampling only occurs in the screen space. Reductions in MSE of up to 166 times were recorded when compared to path tracing.

Lin et al. introduce generalised resampled importance sampling (GRIS)[73], which extends and corrects ReSTIR by enabling RIS to handle correlated samples. Although ReSTIR is unbiased, sample reuse can cause it to converge to incorrect results. The authors develop a theoretical framework with specific criteria to ensure proper convergence, including the use of shift mappings to reuse samples across different domains (e.g., pixels) and a hybrid shift mapping strategy for improved performance. They also highlight the importance of capping the reservoir size M_r (M-cap) to limit reuse; without this cap, results may converge incorrectly. The paper provides ReSTIR PT for experimentation and suggests that GRIS could benefit advanced techniques like manifold exploration. While ReSTIR PT can reuse some caustic paths, caustics remain challenging and may lead to undersampling in both GRIS and ReSTIR.

Zhang and Wang [74] introduced World-Space Spatiotemporal Path Resampling for Path Tracing. Unlike ReSTIR GI, this method is not limited to sampling path samples in screen space. It can sample paths in world-space and can reuse sample points in path samples subsequent to glossy surfaces interaction as well. After sample generation, a hash grid is used to store and query the samples efficiently. The hash-key is based both on position and surface normal, and a cascaded hash-grid is used (not uniform). For resampling, the whole path (generalised path samples) is reused by visible points in contrast to ReSTIR GI where only sample points next to visible points can be connected for reuse. This method demonstrates a 16.6% to 33.1% improvement over ReSTIR GI with regards to noise reduction costing only 4.41% to 8.36% more with respect to speed.

3.6 Other Methods

Moreau & Doggett [75] propose a novel algorithm for rendering indirectly visible caustics in real-time, overcoming limitations of previous screen-space techniques. It works by leveraging hardware-accelerated ray tracing in modern GPUs and first uses path tracing starting from the camera until a diffuse surface is found (collection points). Then, a bounding volume hierarchy (BVH) is constructed around multiple collection points in world space. Light tracing is used to accumulate light around these collection points. This allows caustics to be rendered anywhere in the scene, not just in screen space. The approach ensures consistent caustic rendering even in complex scenarios

involving mirrors and semi-transparent objects. Temporal reuse is also possible, and is implemented using an exponential moving average, giving more weight to newer contributions. A number of scenes were evaluated with a number of dynamic components, such as moving camera, moving lights and animated geometry. The total time for each frame hovered around 25ms per frame (around 40 FPS). However, Bistro was tested with most emissive primitives turned off and dropped to 21-25 FPS. Future work includes refitting the BVH to enhance performance, including motion vectors for caustics to reduce ghosting and using reservoirs to include ReSTIR even on surfaces visible via mirrors.

Fan et al. introduce Specular Polynomials [76], a deterministic (not stochastic) alternative to Newton iterations based methods for finding specular paths. They formulate univariate polynomials encoding multiple specular bounces and their constraints together. Root finding is easier to perform on univariate polynomials. While the method is theoretically unbiased, a first-order approximation for square roots (coming from refraction components) is performed which may lead to inaccuracy. However, for such cases, the authors fallback to refinement using Newton iterations to gain back the accuracy. Additionally, bisection is used in contrast to a comprehensive solver (eigenvalue based) which can lead to finding fewer solutions but bisection is much faster. Furthermore, long-specular chains were not tackled, but since the number of solutions increases dramatically, the authors suggest to combine this with a stochastic approach in order to explore all solutions.

3.7 Related Work

This section reviews work in the specific domains supporting the contributions of this thesis. We examine debugging tools for rendering pipelines, real-time caustics techniques, evaluation frameworks, and denoising methods, contextualising the development of Anvil, CandelaDXR, Forge, and our spectral denoising pipeline.

3.7.1 Debugging Tools for Rendering Pipelines

Total Recall [77] is a debugging framework specifically designed for GPU debugging. It operates by observing the application programming interface (API) calls made by the application to the rendering API and replaying them during debugging. Total Recall enables full execution history by constructing a buffer dependency graph to obtain shader input values. Once all data is collected, it is passed to the CPU emulator for analysis. Additionally, hardware acceleration can be used to speed up the emulation.

In [78], Hou et al. implement an automatic dataflow recording and visualisation debugger. This work is targeted primarily at GPGPU applications. The authors extend

the CUDA language to provide a mechanism whereby the GPU can interrupt the CPU. This is achieved by passing the user shader into a compiler that generates a new shader containing instrumentation code. The advantage of this method is that debugging is performed essentially on the GPU, which means the original code remains faithful to the architecture. For example, it is sometimes the case that floating-point operations on the GPU do not implement the same standard as the CPU. Therefore, debuggers that replay or emulate the GPU on the host tend to suffer from such inconsistencies.

The Ray Tracing Visualisation Toolkit (rtVTK) [79] is a tool that supports the visualisation and analysis of ray-based rendering algorithms. Applications send data to rtVTK via the *rt* API, where ray data such as origin, direction, and length are recorded. Visualisation plugins are subsequently developed to consume this data. Gribble et al. demonstrate this using a ray visualiser plugin, amongst others such as a BVH viewer. rtVTK is designed to help debug issues in ray-tracing applications and to aid students in better understanding ray-based algorithms. rtVTK provides debugging through visualisation, which can be extended via plugins, however it is limited exclusively to rays. In contrast, Anvil is more flexible, as it has no such limitation and, through its design, achieves a broader range of functionality.

GLDebug [80] is a graphics debugger targeted at OpenGL applications. It is designed to inspect state changes occurring in OpenGL. OpenGL is implemented as a state machine, and third-party libraries might modify its state erroneously. GLDebug is capable of detecting such issues and allows developers to avoid cluttering the code with state queries. It also enables state history comparison, so that a working version can be compared with a faulty version, allowing one to pinpoint issues.

Podila and Zhu developed a tool that visualises data flow transfers between the CPU and the GPU [81]. Their implementation generates a graph illustrating how host variables are connected to shader variables. While current tools can detect errors such as failing to bind required variables, they do not visualise broken data connections, and developers must identify such issues manually by reading the source code.

NVIDIA Nsight is a tool that can be used to analyse the shading pipeline running on the GPU. It displays all the draw calls, allows one to record frames for later replay, and presents statistics to help identify bottlenecks. Microsoft PIX¹ is a tool that offers similar features. cuPrintf is another tool that may be used for simpler debugging scenarios. It enables printf-like functionality within a shader. The Unity² framework also provides similar debugging functionalities, such as the Frame Debugger.

Although the aforementioned options are highly useful for debugging, they do not provide means to visualise more specific, common data structures. Furthermore, in most of these tools, debugging occurs after recording, meaning the application is

¹<https://devblogs.microsoft.com/pix/download/>

²<https://unity.com/>

debugged offline. In contrast, Anvil enables online debugging and the use of ready-made visualisers, which can be shared with other users to visualise and debug higher-level data structures.

3.7.2 Caustics Rendering

In [82], Yang and Ouyang claim that the most efficient method for simulating caustics in real time is photon mapping. They present two real-time ray tracing techniques for rendering caustics effects that are low cost and fully interactive, one for metallic and transparent materials, and the other specialised for water surfaces. The former is termed Adaptive Anisotropic Photon Scattering (AAPS), while the latter is termed Ray-Guided Water Caustics (RGWC).

In AAPS, techniques from photon splatting are employed. The algorithm utilises several buffers, two of which control photon emission, namely the task and the feedback buffers. Emitted photons are guided by the task buffer, and their footprint is calculated using photon differentials during hit-bounce time. The photons are subsequently stored in the photon buffer. Each recorded photon is splatted onto the caustics buffer in an elliptical shape, the size of which depends on the scene depth, the surface's normal, and the initial footprint. Finally, the caustics buffer is applied to the scene. The feedback buffer is then looked up and compared with the previous feedback buffer to generate the task buffer. The feedback buffer, of which there is one per light source, is what renders the algorithm adaptive. The average projected area of photons in screen space, together with the variance generated via the screen-space pixel, are used to generate an entry in the feedback buffer. Effectively, each photon is traced back to its emitting light source so that these contributions are accounted for.

The results show that the cost of rendering caustics is between 1 and 2 ms for enthusiast GPUs and 3 to 4 ms for a consumer-grade GPU (RTX 2060). These results are made possible through the use of the RTX cores residing in these GPUs. However, the photon tracing and photon scattering steps still dominated the process, with the former taking approximately 45 ms.

Yang and Ouyang state that several limitations remain. Photon tracing assumes that surfaces are not rough, and dispersion is not computed using continuous spectrum data, which limits its accuracy. Area lights are not optically accurate, and reflected and refracted caustics are generated in separate GPU threads, thereby doubling the cost of photon tracing.

In RGWC, the authors focus exclusively on water surfaces. First, they explain the basic method, where a caustics map is used per light source and the positions are recorded along with the normals of the water surface. Then, reflection and refraction rays are generated from the recorded positions, and once they hit the scene, the infor-

mation of the hit points is also recorded. At this point, caustics are rendered into the screen-space caustics buffer, denoised, and rendered onto the screen.

Yang and Ouyang improve the aforementioned steps in the ray generation phase and the caustics buffer rendering phase by using photon difference scattering (PDS) and procedural caustic mesh (PCM). PDS differs from photon differentials in that no perturbations are applied to the photon in question, instead, data from nearby photons are used to compute the photon coverage. Once the coverage is attained, a quad or elliptical sprite, aligned with the photon's direction and with similar coverage, is smeared onto the receiving surface and finally projected onto the screen. The authors claim that this technique is easier to implement and requires fewer ray casts.

Alternatively, PCM can be used, where hit points are converted into an intermediate mesh. The hit points form triangle lists in the map. The compute shader then interprets the size of each triangle as the photon footprint. The mesh is subsequently rendered to the caustics buffer using a traditional rasterisation pass.

The authors discuss that PCM produces sharp caustics even when the caustics map resolution is low. PCM is better suited for larger areas, such as lakes and the sea, while PDS is optimal for more confined areas, such as a swimming pool. Results indicate that PCM is faster than PDS by approximately 1 to 2.3 ms.

3.7.3 Evaluation Framework

Previous research, such as realXtend [83], introduced a modular approach to virtual world architectures based on the Entity-Component-Attribute (ECA) model. RealXtend emphasised flexibility and network synchronisation, combining an ECS architecture with efficient real-time communication across clients. It allowed for dynamic component addition and real-time updates, facilitating the creation of interactive and extensible virtual environments. Dahl et al. [84] extended this work, implementing WebSocket synchronisation to communicate with web-based clients, illustrating the potential of ECS-based systems for distributed environments. However, while realXtend focuses on flexible and extensible virtual world architectures, its goals differ from those of Forge, which is primarily designed to standardise and streamline the evaluation of rendering systems rather than supporting virtual world simulations.

Elements [85] provides another example of an ECS-based framework, but with a focus on education. It is a lightweight, open-source Python tool designed to teach computer graphics concepts by implementing an ECS architecture within a scene graph. Elements enables students to explore modern computer graphics (CG) pipelines, bridging theory and practice. Although it shares a similar architectural foundation with Forge, the primary use case of Elements is pedagogical, helping students understand CG principles, whereas Forge focuses on evaluating, debugging, and standardising ren-

dering techniques, with support for distributed computation and performance assessments.

NVIDIA’s Falcor [86] provides a platform for real-time rendering research by offering a high-level abstraction over modern graphics APIs such as DirectX 12 and Vulkan. Falcor enables rapid prototyping of advanced rendering techniques, featuring a flexible render graph system, built-in profiling tools, and support for shader hot reloading. However, while Falcor accelerates research through quick composition of rendering pipelines, it does not address the need for standardised evaluations across different rendering engines and lacks built-in network capabilities, limiting its ability to facilitate distributed or remote evaluations. Furthermore, it only provides access to FLIP [39, 40], which specialises in detecting perceptual differences in rendered images, and MSE, which measures pixel-level differences between images.

More broadly, there has been a growing focus on replicability and reproducibility. Bonneel et al. [87] emphasise the importance of making research code available to ensure that results can be replicated. Despite these efforts, much of the available research code remains specific to individual studies or tools, rather than forming part of a reusable, modular framework. While current initiatives focus on improving replicability within specific projects, they do not provide a generalised, system-agnostic platform for evaluating and comparing rendering techniques across different engines and configurations.

To the best of our knowledge, no existing tool fully addresses the niche that Forge aims to fill: a standardised, modular framework for evaluating rendering techniques across diverse pipelines. This may be because most graphics engines are typically evaluated using custom-built tools tailored to specific engines and rarely shared publicly. This fragmentation has led to a lack of standardised tools for cross-engine comparisons, which motivates the development of Forge.

3.7.4 Spectral Rendering and Denoising

Overbeck et al. [88] introduce an adaptive rendering algorithm, using wavelets and variance to sample salient areas, which drastically reduces the sample count needed to achieve lower noise. Levin et al. [89] discuss extending the depth of field in photography through computational imaging, focusing on the frequency power spectrum of the defocus kernel to improve deconvolution quality. The dimensionality of the computation decreases from 4D to 3D in the frequency domain. Egan et al. [90] present a frequency analysis of motion-blurred scenes, including moving objects, specular reflections, and shadows. A novel sheared reconstruction filter aligned to the direction of motion is introduced, allowing for high-quality rendering with fewer samples. Similarly, Egan et al. [91] present a technique for efficiently rendering soft shadows from

complex occluders using a 4D cache and sheared reconstruction filter. Belcour et al. [92] discuss the acceleration of global illumination methods by analysing local light fields in the Fourier domain to reveal illumination consistency in participating media.

The landscape of image denoising has been significantly shaped by advanced transform-based techniques. Starck et al. [93] introduced novel digital implementations of the ridgelet and curvelet transforms, demonstrating their superiority in handling high-frequency components and preserving edge details over traditional wavelet methods in image denoising. Their empirical results offer encouraging evidence for the potential of these methods in complex image reconstruction tasks, including caustics. Similarly, Kamble et al. [94] conducted a thorough performance evaluation of image denoising algorithms using wavelet, ridgelet, curvelet, and contourlet transforms. They proposed a new robust performance measure, which further elucidates the strengths and limitations of these techniques in preserving image quality during denoising processes. Ansari and Budhhiraju [95] evaluate these transforms against contourlet transforms on remotely sensed images, with curvelets showing superior edge preservation. In a more focused study, Yu et al. [96] developed a curvelet-based denoising method that incorporates polynomial interpolation and adaptive thresholding, specifically enhancing the retention of textural details in medical imaging, which is critical for telemedicine applications. Halidou et al. [97] review wavelet-based denoising algorithms, distinguishing BM3DM_DWT as the most effective for its simplicity and efficiency in various tests, including standard MATLAB images. Their comparative analysis, grounded on metrics such as PSNR and SSIM, provides a nuanced understanding of how different wavelet methods perform across typical image denoising scenarios.

For a comprehensive review of image denoising techniques using deep learning, Jebur et al. [98] provide an extensive overview of various methodologies and their applications in the field. Their study covers a broad range of methodologies, comparing machine learning models, convolutional neural networks (CNNs) and hybrid approaches, and offers insights into the progression and challenges of deep learning techniques in image denoising. To the best of our knowledge, no comparative study focuses directly on the denoising of caustics in PBR-generated images.

3.8 Summary

This chapter has surveyed the historical and contemporary landscape of light transport simulation, critically evaluating the progression from foundational unbiased Monte Carlo methods like path and bidirectional path tracing to the biased but efficient photon mapping and radiosity techniques. The analysis identified that while modern offline methods such as SMS offer sophisticated solutions for complex caustic paths, and real-

time resampling frameworks like ReSTIR provide transformative performance for general global illumination, a significant gap remains for high-performance, GPU-amenable algorithms capable of rendering high-frequency caustics. The review further contextualised the thesis contributions by examining related work in debugging tools, specialised caustics rendering, evaluation frameworks, and spectral denoising. Ultimately, this review underscores that while significant advances have been made in both offline accuracy and real-time performance, the synthesis of these domains into a dedicated solution for caustics remains an open and critical challenge for real-time rendering.

4 Anvil

Offline rendering refers to image synthesis performed without real-time constraints, with its origins dating back to the 1960s. At that time, rendering models lacked full physical accuracy, though foundational work was gradually advancing towards physically based approaches. Over the ensuing decades, physically based rendering became attainable, yet remained largely confined to offline applications until recent years. Throughout this period, numerous algorithms have been developed to address global illumination, each exhibiting varying strengths depending on scene configuration. No single solution universally accommodates all scenarios; instead, specialised methods have emerged to solve specific challenges. Many unbiased techniques employ stochastic sampling, as this remains the only demonstrably scalable approach to solving the rendering equation. Ultimately, computational efficiency has persisted as a central challenge. However, with modern GPUs now incorporating hardware-accelerated support for ray-based methods, certain traditionally offline techniques have become feasible for real-time applications.

The primary objective of this thesis is the real-time generation of caustics by adapting offline, unbiased rendering techniques for GPU execution. However, migrating such methods to the GPU presents significant challenges due to fundamental architectural and execution model disparities. While CPUs are optimised for sequential processing and branch-heavy code, leveraging deep cache hierarchies, GPUs thrive when handling massively parallel, data-aligned workloads across thousands of lightweight threads.

4.1 Initial Methodology

To facilitate this adaptation, we begin with a path tracer, as it represents one of the most straightforward ray-based methods well-suited to GPU implementation. In its purest form, path tracing minimises branching, relying primarily on Monte Carlo integration which is an intrinsic requirement of virtually all stochastic rendering algorithms. Furthermore, path tracing is stateless, eliminating the need to store intermediate data; only immediate values, typically held in GPU registers, are retained. Crucially, path tracing is inherently unbiased, and this is a property we aim to preserve while also natively supporting global illumination.

To achieve this objective, we selected Candela [14], a CPU-based rendering engine. Candela is an ideal candidate due to its clean, modular design, which adheres to robust object-oriented principles. While alternative engines, such as PBRT, are more complex and architecturally optimised for their specific use cases (e.g., through ab-

stractions like spectral rendering support, which requires generalising over radiance primitives), our focus resides in a narrower domain: namely, caustics rendering with radiance derived exclusively from RGB channels.

PBRT’s intricate design accommodates a broad range of features, including diverse shaders, geometric primitives, and spectral operations. This flexibility introduces overhead for our targeted application. In contrast, Candela’s streamlined architecture avoids such complexity, enabling straightforward modifications and extensions tailored to our needs.

4.2 Path Tracer Verification

Candela offers a clean, well-structured framework as our foundation, complete with an existing path tracer implementation. However, verifying the correctness of this implementation is critically important at this stage, as the rendered images produced by this path tracer will serve as our reference for equivalence testing with subsequently developed shaders.

For our initial verification phase, we employed the canonical Cornell Box scene, a standard test case for evaluating global illumination algorithms. This benchmark scene features an enclosed room illuminated by a planar light source containing two cuboid objects, with all surfaces exhibiting purely diffuse reflectance. The scene’s geometrical simplicity and well-understood light transport characteristics make it particularly suitable for verifying fundamental shader implementations.

Simple visual inspection confirmed the Cornell Box scene converged to the physically plausible reference solution. While standard image metrics (e.g., SSIM) are typically employed for such comparisons, we initially opted for direct pixel value inspection via a lightweight diagnostic tool. This tool, implemented using web technologies for rapid prototyping, displayed raw radiance values at cursor positions, enabling efficient sanity checks. The choice of web technologies (e.g., Node.js libraries) afforded both simplicity and extensibility, though the basic implementation proved sufficient for our immediate needs. Crucially, this facilitated verification of scene configurations, such as confirming emitted radiance values matched the Cornell Box’s light source specifications.

Candela’s next event estimation (NEE) path tracing variant exhibits faster convergence while theoretically producing identical results (within floating-point tolerances). Our pixel inspection tool enabled efficient pairwise comparison of images from both renderers. For stricter verification, we subsequently developed additional tooling to compute standard error by sampling per-frame radiance values. This required minor renderer modifications to suppress output volume, typically restricting data collection

to individual pixels or small image regions.

4.3 Debugging Challenges

During extended testing, we observed sporadic black pixels indicating discontinuities. Initial radiance inspection revealed anomalously low values in localised regions, contradicting expectations for a purely diffuse environment. Conventional debugging proved ineffective until we developed a ray visualisation tool. This immediately revealed the root cause: a self-occlusion artefact occurring at specific grazing angles.

Upon completing the verification of Candela’s path tracer, we proceeded with its GPU implementation, namely CandelaDXR. During evaluation, we observed significant firefly artefacts appearing exclusively when rotating the area light within the Cornell Box scene. Initial debugging proved ineffective in identifying the root cause of these high-variance pixels.

To investigate further, we instrumented the GPU shader to capture full path information for specified pixels and frames via DirectX read-back buffers. We subsequently extended our visualisation tool to reconstruct complete light paths, displaying radiance values at each path vertex. This revealed an illuminating pattern: the fireflies consistently occurred when shaded points approached extremely close proximities to light sources.

The visualisation demonstrated this was a mathematical singularity arising from the inverse square term ($1/r^2$) in the illumination calculation. As the distance r between the shaded point and light source diminishes, this term grows exponentially, causing numerical instability. We determined this artefact could be effectively mitigated by implementing an alternative Monte Carlo sampling strategy for near-field light interactions, particularly when r approaches zero.

4.4 Introducing Anvil

Throughout this development effort, we recognised that debugging and diagnosing issues in computer graphics applications presents significant challenges, particularly when transitioning from CPU to GPU-based implementations. While traditional CPU renderers already exhibit considerable complexity in debugging, GPU environments exacerbate these difficulties due to their distinct architectural paradigms and lack of fundamental debugging primitives. As our work progressed, we observed that the tools we developed could be generalised beyond path tracing to address other computational domains, including rasterisation and GPGPU computing.

To streamline the adaptation of offline rendering techniques for GPU execution, we consolidated our existing diagnostic tools into a unified framework: Anvil. This visual debugging and analysis platform is designed to integrate seamlessly with user applications, enabling high-level inspection of complex rendering behaviour. Anvil prioritises three core principles:

- **Efficiency:** Operating within the constraints of interactive debugging while maintaining real-time responsiveness.
- **Extensibility:** Supporting custom visualisations and modifications to existing analysis tools through a modular architecture.
- **Flexibility:** Applicability beyond computer graphics to domains requiring high-level debugging abstractions.

A key innovation of Anvil lies in its structured representation of 3D graphics primitives through atoms (basic elements, such as vectors) and molecules (compound structures, such as rays). These abstractions provide reusable building blocks for visualisation, while remaining extensible for domain-specific customisations. By formalising these conceptual units, Anvil accelerates debugging workflows for both common rendering tasks and specialised GPU development efforts.

4.5 Design Principles

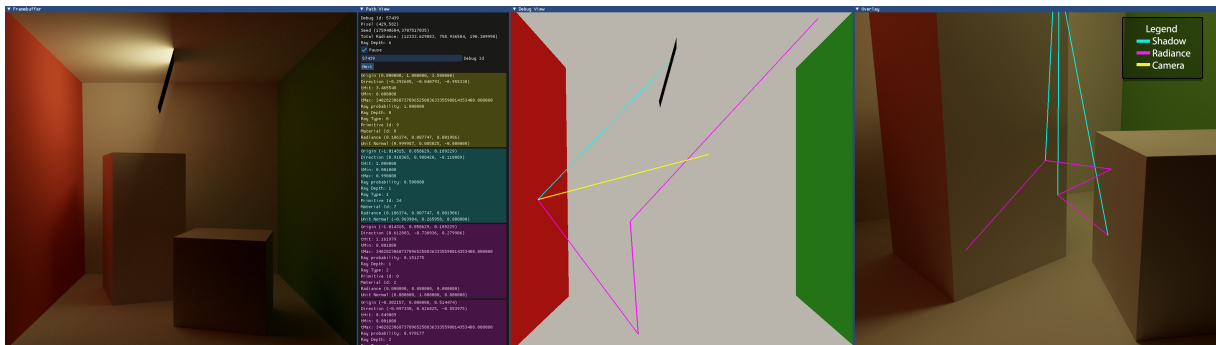


Figure 4.1 Anvil debugging session. From left to right: Application framebuffer view, path view, debug view, overlay

One important design principle adopted in Anvil is simplicity (see Figure 4.1). Anvil is a tool designed to help analyse and debug graphics applications; it is important for end users to remain focused on their current task and not waste time configuring an external tool. This is why we strongly believe Anvil should be simple to set up and use. Reflection, which is the ability of a program to inspect and modify its own structure and

behaviour at runtime, together with macros have been implemented to make interfacing with Anvil as simple as possible.

Anvil was designed to promote component reusability. It can be divided into two functional groups: instrumentation, and analysis and visualisation. While instrumentation is tightly coupled to the user's code, the analysis and visualisation components are independent of it. This enables any visualisation and analysis logic to be reused.

Anvil is data-driven and was designed primarily to use the ECS design pattern. This pattern was chosen because it provides Anvil with the necessary flexibility. It allows Anvil to decouple data from logic. Since the user is expected to post a continuous stream of data seamlessly, this design is a natural fit. A key advantage is that it helps avoid complex and deep inheritance hierarchies. Without an ECS-like design, an entity might need to implement multiple interfaces, where each interface implementation and its associated data would represent a component in ECS.

Anvil is designed to be cross-platform and modular to address flexibility. It is not tightly coupled to the graphics domain and is API agnostic. This loose coupling enables it to run on any platform with minimal dependencies. A plugin system addresses its extensibility aims. Finally, Anvil uses reflection, a concept not natively present in C++, to understand and visualise user data structures.

4.6 Design

Figure 4.2 illustrates the high-level architecture of Anvil. Atoms and molecules (Section 4.4) are used during the application's instrumentation phase. Both the required visualisation and analysis depend on this instrumentation. Systems communicate via events. The registry is used throughout Anvil to store atom and molecule definitions, which are subsequently used by the application and its systems. Anvil is designed for use across different hardware architectures and APIs, such as OpenGL and DirectX.

In an ECS context, an entity is the object we wish to debug. A component is a field or method within that object, and a system is a debugger. Anvil provides several systems out of the box, including a camera visualiser and a mesh visualiser. In Anvil, reflection is used to register and categorise class members as components. Systems can then operate on these components as needed. For example, a wireframe viewer system could iterate through all mesh entities possessing a position component to render a wireframe of a scene.

An atom is the smallest unit of data upon which Anvil operates. Atoms also have semantic attributes attached to them. For example, although a position and a direction can both be represented by vectors with the same data structure, they possess different meanings. An atom is therefore tagged to preserve its semantics. A molecule

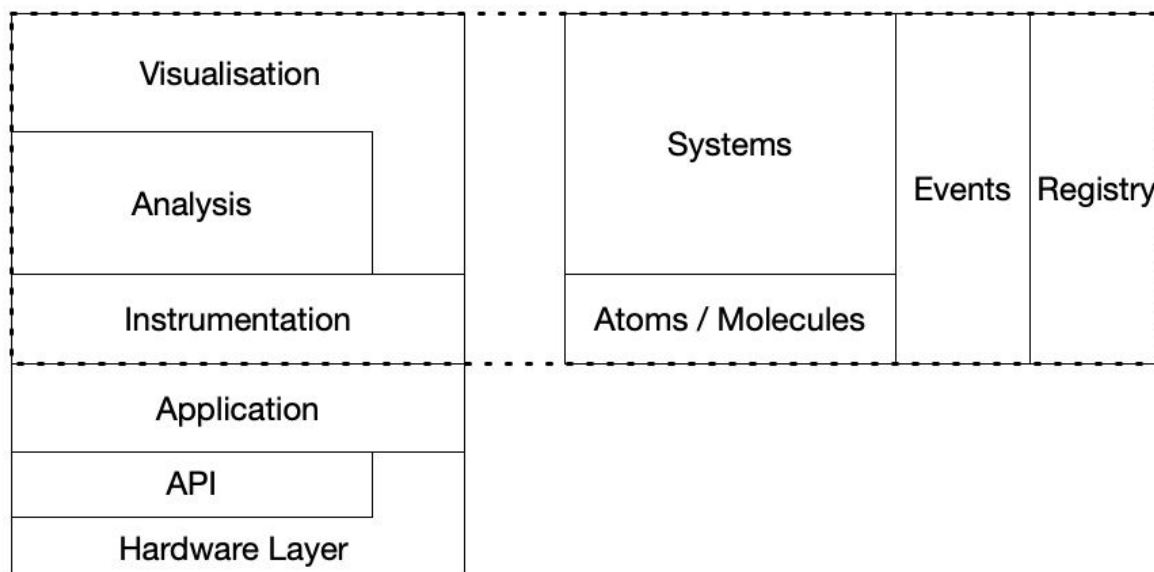


Figure 4.2 Anvil architecture

in Anvil is a concept that ties multiple atoms together, the aggregation of which forms new semantics. For instance, a ray is a molecule composed of a position atom and a direction atom.

4.6.1 Reflection

To register an atom or molecule, the user’s data structure must first be registered. For example, a position vector contains three floating point numbers, named x , y , and z respectively. Anvil can only interpret the user’s data structure if semantics, or tags, are applied to it. Reflection is a technique that can simplify this process. However, as C++ does not yet support reflection, some degree of manual mapping is hard to avoid.

One option considered was to parse the user’s code using CLang’s parser and register all metadata using the information obtained. Nevertheless, the user would still need to manually assign semantics to the fields, as the data structure members may not have standardised names. We concluded that this option would only be viable if accompanied by a tool allowing the user to select and tag fields visually via a graphical user interface (GUI). However, before committing to further development, it was decided that the first version of Anvil would operate with manual input so that the concept could be evaluated first.

Existing reflection libraries are fully featured and not designed solely for simple scenarios. These libraries are large, and at the time of evaluation, their performance was not satisfactory. Since Anvil requires only a very small subset of these features, namely reading and writing fields and calling getter and setter methods, it was determined that implementing a minimal reflection library from scratch was the best approach for

the time being. This library was implemented using a C++ feature called template metaprogramming. This feature enables the extraction of class metadata, allowing data to be queried at runtime.

4.6.2 Compatible Members

The developed reflection library provides mechanisms to register data members, getter and setter methods, and wrapper functions. The `Point2D` class in Table 4.1 defines a simple two-dimensional point. This point consists of two floating-point coordinates; the x coordinate is tagged as “X” and the y coordinate is tagged as “Y”.

The first row in Table 4.1 shows the simplest case for registering a user data structure. All required fields are public, and the user can simply register those member fields in an external C++ file. The second row shows the case where the data members are private but public getter methods exist. Here, the user can register the public methods. Note that these public members return a pointer to the actual data. While this is not strictly necessary, it allows systems and visualisers to modify the data. The third row illustrates a scenario where, although the data is public, it is not in the required format, being stored in an array. Consequently, a wrapper class must be implemented. This class stores a reference to the two-dimensional point and defines getter methods for the x and y coordinates. The registration code then interfaces with this wrapper class. Finally, Table 4.2 presents the case where the required data is private. Here, the user must modify the class definition by adding the `REFLECT` macro. Internally, this macro uses the C++ *friend* keyword to grant the registration code access to the private fields.

Reflection is responsible for managing the user objects that require observation. Through a user-provided mapping, this component extracts data from the user object and transforms it into a stream of atoms and molecules. This stream is the fundamental data upon which Anvil is designed to operate.

4.7 Visualisation and Analysis

Visualisers and analysers are implemented as systems within the ECS design pattern. They operate directly on the stream of atoms and molecules and are designed to work only with specific types. For example, a path visualiser can interpret a path molecule but will not process a ray molecule. Furthermore, molecules can be decomposed into their constituent atoms, which can then be reinjected into the stream. This enables other systems to operate on these atoms if necessary. This functionality is provided by the built-in Decomposer system.

Anvil provides a straightforward interface for users to implement their own visualisers or analysers. At a minimum, the user must implement the `System` interface.

User Data Structure	Registration Code
<pre>class Point2D { public: float x, y; };</pre>	<pre>#include "reflection/reflect.h" REFLECT_BEGIN(Point2D); addMember("X", &BaseType::x); addMember("Y", &BaseType::y); REFLECT_END();</pre>
<pre>class Point2D { private: float x, y; public: float* getX(); float* getY(); };</pre>	<pre>#include "reflection/reflect.h" REFLECT_BEGIN(Point2D); addMember("X", &BaseType::getX); addMember("Y", &BaseType::getY); REFLECT_END();</pre>
<pre>class Point2D { public: float xy[2]; }; class Point2DWrapper { private: const Point2D& ref; public: Point2DWrapper(const Point2D& ref) : ref(ref) {} float getX() const { return ref.xy[0]; } float getY() const { return ref.xy[1]; } };</pre>	<pre>#include "reflection/reflect.h" REFLECT_BEGIN(Point2DWrapper); addMember("X", &BaseType::getX); addMember("Y", &BaseType::getY); REFLECT_END();</pre>

Table 4.1 Registrations

User Data Structure	Registration Code
<pre>#include "reflection/reflect.h" class Point2D { private: float x, y; REFLECT(Point2D); };</pre>	<pre>#include "reflection/reflect.h" REFLECT_BEGIN(Point2D); addMember("X", &BaseType::x); addMember("Y", &BaseType::y); REFLECT_END();</pre>

Table 4.2 Registrations of private data members

Listing 1 System interface

```

class System
{
public:
    virtual ~System() = default;

    // This method passes the entities to this system
    virtual void execute(std::vector<std::shared_ptr<Entity>>& entities) = 0;
};

```

Listing 2 IDrawable interface

```

class IDrawable
{
public:
    IDrawable() = default;
    virtual ~IDrawable() = default;
    IDrawable(const IRenderer&) = delete;

    // Draw methods
    virtual void draw(IRenderer& graphics) = 0;
    virtual void drawUI() = 0;
    virtual const char* toString() const = 0;
};

```

This interface contains a single method, *execute*, depicted in Listing 1. This method receives a list of entities for processing by the implemented system.

4.7.1 OpenGL Visualisers

The implementation of a basic System provides the user with significant possibilities. For instance, a System can simply export data for use by a third-party application. However, as Anvil is intended to be an interactive analysis tool, some fundamental visualisation capabilities are provided.

An OpenGL renderer is implemented in Anvil. This renderer can draw basic geometry, such as lines and triangles. Therefore, to add a line, the user simply provides two points. Additionally, GUI elements, including buttons and text boxes, are provided through ImGui¹. If a system intends to use the OpenGL renderer to draw in the main Anvil window, in addition to the System interface described in Listing 1, it must implement the interface shown in Listing 2.

The *draw* method is called by the renderer for every frame, and each frame is synchronised with the screen refresh rate. It is used to draw content that is particularly dynamic. The *drawUI* method is called when the system needs to draw a user menu for interaction. These two methods are invoked by the OpenGL renderer thread. This

¹<https://github.com/ocornut/imgui>

thread may be the same as the default main thread if the renderer is initialised without the parallel flag.

The parallel flag is required to preserve user interactivity. It may be the case that the user's renderer, which is being debugged, runs slower than interactive rates. Assuming a speed of 1 FPS, if the same thread is used to drive the OpenGL renderer, then the Anvil window will also run at 1 FPS. This can be too slow for effective visualisation. Therefore, the user is permitted to initialise the OpenGL renderer using a separate thread. It is important to note that when using the parallel flag, the *execute* method is called by the main thread, while the *draw* and *drawUI* methods are called by a separate thread. This means that any shared data in the implementation must be protected, potentially through the use of mutexes.

4.7.2 Example Line Visualiser

Listing 3 shows an example of a basic visualiser that draws line molecules using the OpenGL renderer. Some details have been omitted for brevity. The *execute* method is called each time the Anvil pipeline runs. The program iterates through all entities and processes those possessing a line component. Using the reflection library, the "Start" and "End" molecules are read, representing the start and end points of the line to be drawn. This line is then added to the line mesh. Once all relevant entities have been processed, the renderer is instructed to reload the mesh, and the new lines are drawn. It is important to note that since the content does not change outside of the Anvil pipeline, the *draw* method is not required and is therefore implemented as empty. A simple read-only text element, displaying the size of the mesh, is implemented in the *drawUI* method.

Listing 3 Basic visualiser

```

void MyVisualiser::execute(std::vector<std::shared_ptr<Entity>>& entities) {
    mesh->clear();
    auto lineMeshGroup = make_unique<MeshGroup<Line>>();

    for (auto& entity : entities) {
        for (auto& component : entity->components) {

            // Only process lines
            if (component->name != "Line")
                continue;

            const auto& cTD = component->typeDescriptor;
            auto p1 = readPoint(cTD.getMember("Start"), component->datum);
            auto p2 = readPoint(cTD.getMember("End"), component->datum);

            lineMeshGroup->addShape(p1, p2);
        }
    }

    mesh->groups.push_back(move(lineMeshGroup));
    this->renderer->reloadMesh(mesh);
}

void MyVisualiser::draw(renderer::IRenderer&) {}

void MyVisualiser::drawUI() {
    ImGui::PushID(this);
    ImGui::Begin("My Visualiser");
    ImGui::Text("Mesh size: %u", mesh->size());
    ImGui::End();
    ImGui::PopID();
}

```

4.7.3 Third-party Library Considerations

Anvil supports multiple concurrent systems, each of which can use either the same OpenGL renderer or a different instance. Each instance is visualised in a separate window. Supporting this scenario introduces a challenge: OpenGL operates as a state machine with global state per process. To enable Anvil to manage multiple OpenGL windows, a context switch must occur to save the current window state. This behaviour is achieved using a global round-robin scheduler, where each task (one per window) sets up its current context and executes its individual render loop. Furthermore, employing a single scheduler ensures that one thread handles all OpenGL operations, which

is typically the recommended approach.

A problem arises when using a system with the OpenGL renderer while the user’s rendering engine also uses OpenGL. If Anvil’s OpenGL renderer is initialised with the parallel flag, Anvil makes OpenGL calls from a separate thread, different from the thread used by the user’s rendering engine. This can lead to race conditions, resulting in a corrupted sequence of calls to the OpenGL library. This issue can be mitigated in two ways. One method is to disable the parallel flag when initialising Anvil’s OpenGL renderer. Alternatively, Anvil provides `OPENGL_BEGIN` and `OPENGL_END` macros for the user to wrap any code that calls OpenGL. These macros generate lock and unlock calls to a shared mutex, preventing conflicts between OpenGL calls. While effective, this approach may not be ideal due to the number of code modifications required.

The same issue can occur with the ImGui library. Anvil uses ImGui to display context menus and relevant data, and visualisers may use ImGui without additional setup. Like OpenGL, ImGui contains global state and can only be initialised on a per-window basis. Consequently, if a user employs ImGui as a shared library, conflicts may arise. Anvil’s OpenGL renderer addresses this by implementing a context switching mechanism. Whenever Anvil’s pipeline is active, it saves the user’s ImGui context (if present) and restores it afterwards. When the parallel flag is used, the user must protect ImGui usage with a global mutex object provided by Anvil. Another potential solution is to compile and link ImGui statically into Anvil, though this method has not yet been tested.

4.8 Registry

The registry’s role is to track all registered atoms, molecules, and systems. Using the registry, atoms and molecules can be assigned multiple names. For example, a molecule with position semantics could be labelled as *Position*, *Pos*, and so forth. This allows different visualisers that refer to position using different terminology to work together. Additionally, the registry can be used to register dependencies for visualisers. With this information, a list of visualisers compatible with a given set of atoms or molecules can be queried.

4.9 Pipeline Flow

Anvil is designed to be called from the render loop commonly found in most rendering applications. Users are expected to register the objects they wish to observe and then delegate control to Anvil. At this point, Anvil enumerates all available systems, which then process the user data. Users can communicate directly with these systems via

event callbacks. Figure 4.3 illustrates the data flow within Anvil. Data is mapped via the reflection mapper and transformed into objects containing atoms and molecules. These objects are stored in the Object Pool. Visualisers and other systems can then consume and edit these objects. Furthermore, they can publish events for both internal and external communication.

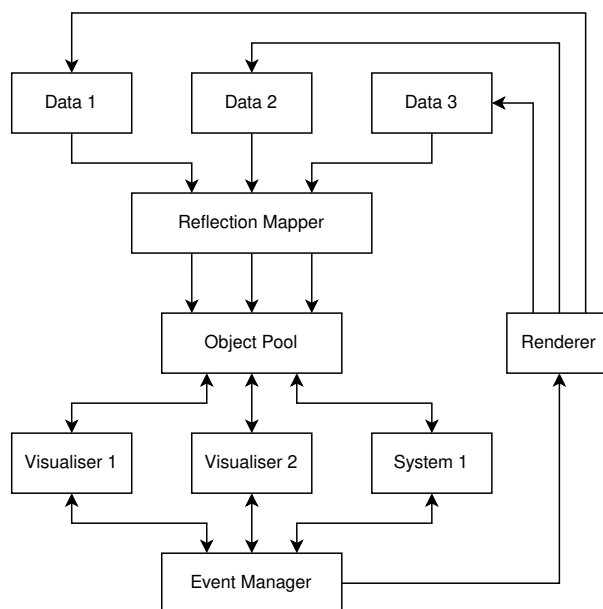


Figure 4.3 Anvil data flow diagram

4.9.1 Initialisation Stage

Anvil cannot be used before it is configured. First, the required atoms and molecules must be registered, which may involve using the reflection library. Subsequently, a list of compatible visualisers can be queried using the registry. The required visualisers must then be initialised. If these visualisers require the OpenGL renderer, the renderer must itself be initialised beforehand.

4.9.2 Runtime Stage

At runtime, a stream of atoms and molecules is produced. Each atom or molecule in the stream represents a debug point within the user application. The Anvil pipeline is then triggered by calling the *tick* method. This enables Anvil to execute the visualisers, which in turn use the reflection library to process the published stream of atoms and molecules. Visualisers may emit events to interact with the user or with each other. When this cycle completes, the *tick* method returns, returning control to the user. The user can then modify the previously published stream or publish an entirely new one,

and the cycle continues until the session concludes. A more practical example is provided in Section 4.10.

4.10 Typical Usage Example

The data requiring monitoring must first be identified. Subsequently, the relevant atoms and molecules must be loaded and registered. If a required atom or molecule does not exist, it must first be developed. The next step is to instrument the application. This involves mapping the atoms and molecules to the user's data structures and submitting the actual data to Anvil for analysis and visualisation. Listing 4 demonstrates a mapping from the user's *Vector3* class, where the constituents of the atom "X", "Y" and "Z" are mapped to the public class data members *x*, *y* and *z* respectively.

Listing 4 Anvil registrations

```
#include "reflection/reflect.h"

REFLECT_BEGIN(Vector3);
addMember("X", &Vector3::x);
addMember("Y", &Vector3::y);
addMember("Z", &Vector3::z);
REFLECT_END();
```

If data members are private but public getter methods exist, these can be referenced instead. If no public getter methods are available, the user must employ the `REFLECT()` macro. This macro designates the relevant reflection class as a friend of the target class; in C++, a friend class can access the private members of the class granting friendship. Listing 5 demonstrates the usage of this macro. If the user cannot modify the class requiring mapping, and this class exposes its private data through other means, a wrapper class must be used. The user can create a wrapper class that exposes the data via getter methods.

Listing 5 Handling private data members

```
#include "reflection/reflect.h"

class Vector3 {
    float x, y, z;

    REFLECT(Vector3);
};
```

The final instrumentation step is to submit the actual data. This is performed using the *addReflectionEntity* method. For example, to monitor a ray, the user would call `Anvil::addReflectionEntity("Ray", ray)`, where *ray* is the user's ray instance and the string "Ray" represents the molecule semantics. For debugging GPU-based applications, an additional step of copying the relevant data from the shader to the host is required.

The user should then identify the type of analysis needed. This determines which system to use. The list of available and compatible systems can be queried from the registry using the previously identified atoms and molecules. If the desired system is not available, it must be developed and registered within the global registry. Systems can be developed easily as they only need to implement an *execute* method.

Events in Anvil are used to propagate messages between systems and provide feedback to the user application. Events can be used to dispatch messages related to breakpoints as well as system-related data. Finally, whenever analysis and visualisation are required, the user calls `Anvil::tick()`.

4.11 Evaluation

Anvil was evaluated on three problems in the context of real-time physically based rendering. This section presents the results of our evaluation and compares them with traditional debugging approaches.

4.11.1 Path Tracing

In path tracing, Monte Carlo methods are typically used to compute an approximate solution to the rendering equation. Within these stochastic methods, reducing variance accelerates convergence to the solution. One such variance-reduction technique is NEE.

Figure 4.4 shows the Cornell Box scene with an area light source directed towards the ceiling. Numerous very bright pixels, known as *fireflies*, were observed. The fireflies persisted even after computing 50000 samples per pixel, but would slowly fade away after many more samples. It was not immediately clear whether this was an implementation issue or a normal occurrence when using NEE.

Traditional Approach

NVIDIA Nsight was used to debug this problem. The first step involved reading the radiance values of the affected pixels. It was noted that these values were abnormally high. Even though a large number of radiance samples were taken, the average remained elevated due to a single sample causing a significant spike.

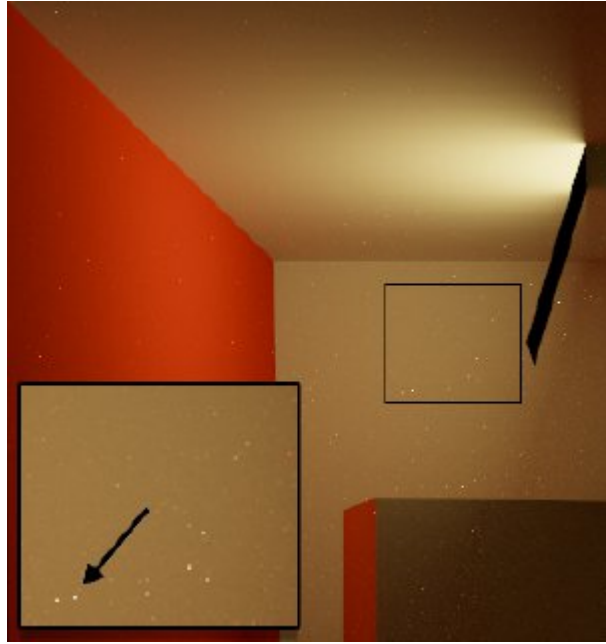


Figure 4.4 Rendering artefacts (fireflies) persisting after 50K samples

The next step was to capture a ray that generated this phenomenon. The strategy employed was to replicate the issue and capture the seed of the pseudo-random number generator (PRNG). The application was modified to allow the PRNG to be seeded. As shader debugging is limited in DirectX 12, the only option to capture the required data was to copy it from the GPU to the host in every frame. With the application seeded, the pixel generating the issue was known in advance, enabling ray data to be captured specifically for that pixel. Whenever the radiance value exceeded a preset threshold, the path data for that particular frame and pixel were saved for offline analysis.

The data were then analysed by meticulously examining the path data. To visualise the information, the rays were manually drawn in a third-party modelling application over the Cornell Box scene. Eventually, it was observed that paths collecting radiance close to the light source had extremely high values that propagated back to the root ray of the path.

Anvil Approach

Debugging with Anvil involved the following steps. First, the application was modified to allow PRNG seeding. Additionally, functionality to copy data from the GPU to the host for a specific pixel was implemented.

For this scenario, Anvil's path visualiser was used. The visualiser requires a *path* molecule, so the host data retrieved from the GPU had to be mapped accordingly. Once all necessary mappings were registered, the data for the selected pixel could be sent to Anvil using the host's original data structures, which were tagged with the string

“Path”. Internally, Anvil adds the path molecule to an entity. At the end of the frame, control is then delegated to Anvil.

For this debugging session, a conditional breakpoint was set in the path visualiser to pause execution when the radiance value exceeded a predefined threshold. The renderer was left running until the breakpoint was triggered.

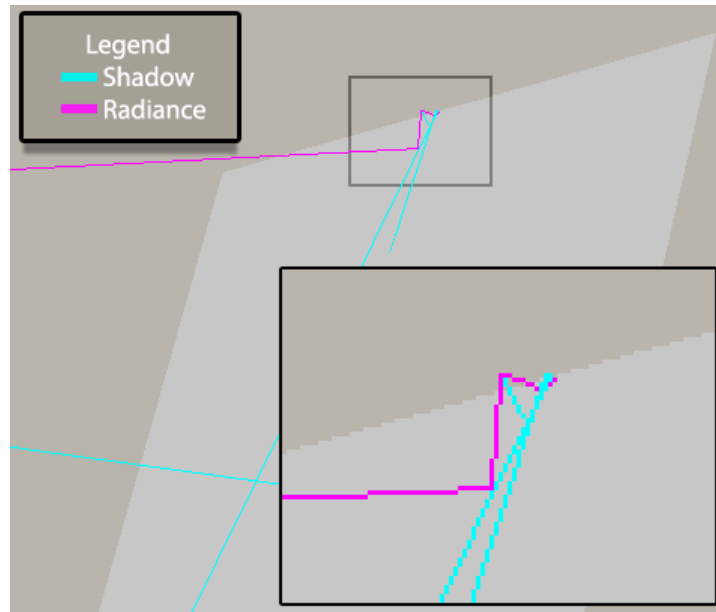


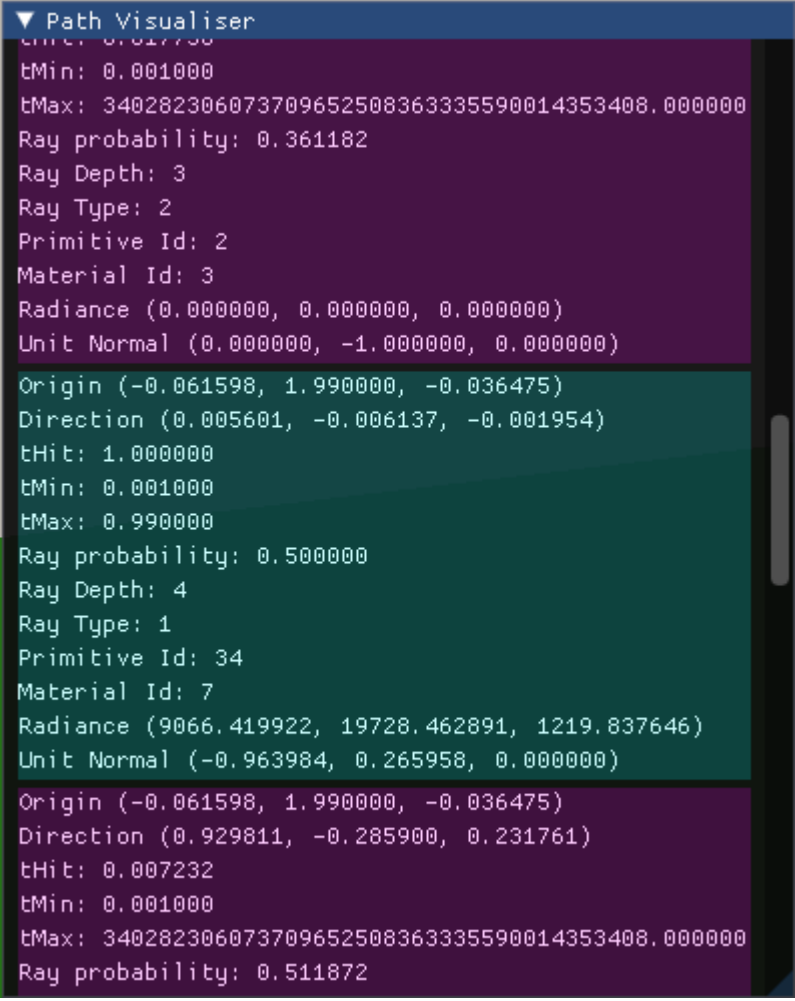
Figure 4.5 Path visualiser focusing on the anomaly

Figure 4.5 shows the paused path visualiser, where the rays for the current frame are displayed alongside the scene geometry. Every hit point is listed in Figure 4.6. The green panel captures the moment when the radiance value became abnormally high. It can be seen that the ray in this case is very short. As this is a shadow ray, the magnitude of the direction vector determines its length. The magnitude was calculated to be 0.0086 units, and the conclusion was that the issue arose from the nature of NEE. When applying the inverse square law, a division by 0.0086 squared is required, resulting in the high radiance value observed.

Discussion

Both approaches share common setup steps. For instance, both methods require copying path data from the GPU to the host. While not strictly necessary, initialising the PRNG with a specific seed is beneficial when analysing individual pixels.

While NVIDIA Nsight is useful, it is not sufficient for this scenario, as data must be saved for offline analysis. In situ debugging does not assist here because visualising the problem using only the vector data available at runtime is difficult. The exported data can be visualised over the existing scene using a third-party application. Visualised rays must then be compared with ray metadata to understand the radiance val-



```

▼ Path Visualiser
tMin: 0.001000
tMax: 340282306073709652508363335590014353408.000000
Ray probability: 0.361182
Ray Depth: 3
Ray Type: 2
Primitive Id: 2
Material Id: 3
Radiance (0.000000, 0.000000, 0.000000)
Unit Normal (0.000000, -1.000000, 0.000000)

Origin (-0.061598, 1.990000, -0.036475)
Direction (0.005601, -0.006137, -0.001954)
tHit: 1.000000
tMin: 0.001000
tMax: 0.990000
Ray probability: 0.500000
Ray Depth: 4
Ray Type: 1
Primitive Id: 34
Material Id: 7
Radiance (9066.419922, 19728.462891, 1219.837646)
Unit Normal (-0.963984, 0.265958, 0.000000)

Origin (-0.061598, 1.990000, -0.036475)
Direction (0.929811, -0.285900, 0.231761)
tHit: 0.007232
tMin: 0.001000
tMax: 340282306073709652508363335590014353408.000000
Ray probability: 0.511872

```

Figure 4.6 Path visualiser showing additional information

ues. Conversely, with Anvil, the data is readily accessible. However, mappings must first be established for Anvil to interpret the data, and Anvil itself must be linked with the application.

Both methods involve setup time, but using Anvil is faster. The traditional method is time-consuming, particularly when exporting data to third-party applications. Furthermore, even with rays visualised, continuous checks are required to determine which ray is collecting a particular amount of radiance. With Anvil, both the ray and its data are instantly available, and the scene can be explored interactively. Furthermore, one can interactively step through successive frames to verify that the NEE algorithm functions correctly when rays are distant from the light source. Verifying this with the traditional method requires repeating the laborious export and import process multiple times.

In conclusion, it was determined that because the calculation involves a divisor dependent on the squared distance from the light source, the radiance value becomes numerically unstable at small distances.

4.11.2 Bidirectional Path Tracing

The bidirectional path tracing algorithm is a technique typically used to render PBR scenes. Sampling a pixel requires initiating paths from both the camera and a light source. The two paths are then connected and weighted appropriately. Our CPU-based implementation follows Lafortune's algorithm [3]. A problem was observed where shading was not correctly applied at the edges of objects. Figure 4.7 shows a close-up of an intersecting edge between the tall box and the floor in the Cornell Box scene. After 4000 samples, the edges remained very noisy. It was unclear whether this was an implementation issue or a characteristic of how Lafortune's algorithm operates in this context.

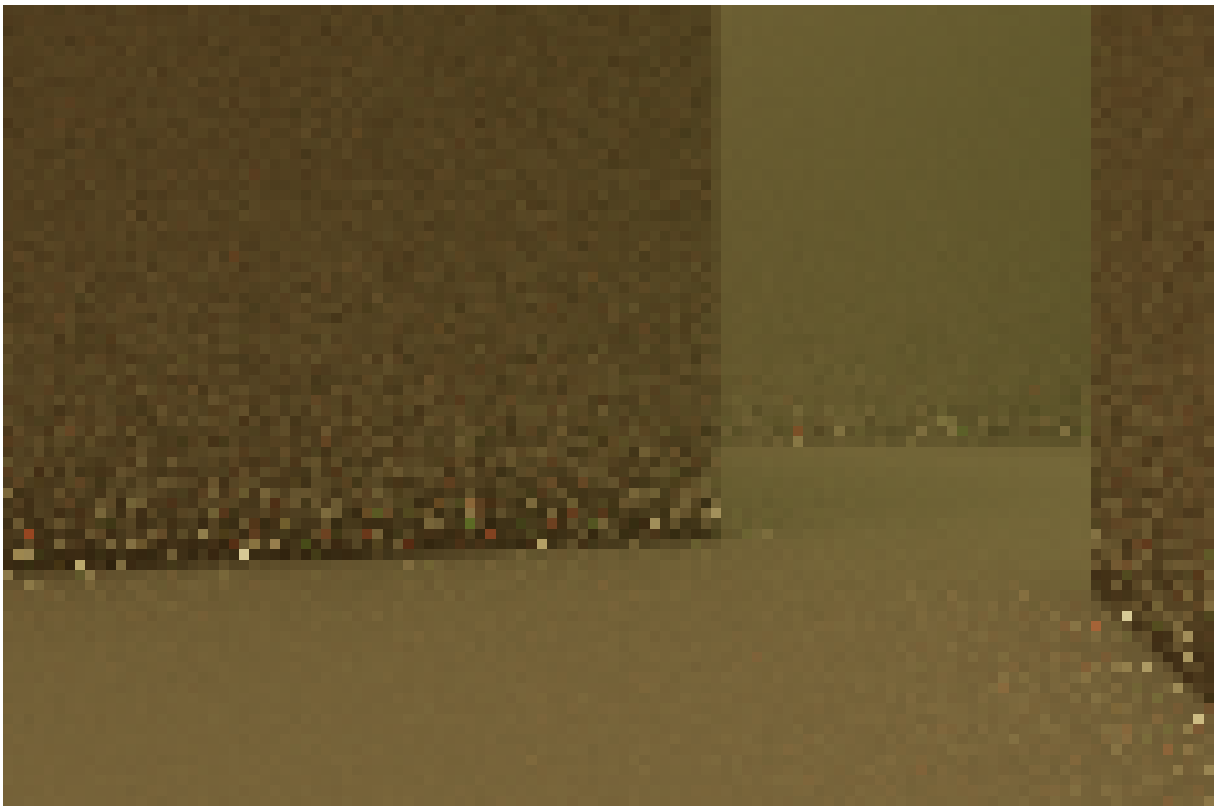


Figure 4.7 Rendering artefacts persisting after 4K samples

Traditional Approach

A standard debugging session was conducted. Breakpoints were placed at locations where radiance contributions are accumulated to verify whether all pixels in the region were being sampled, which was confirmed. A second test was performed to check for the presence of NaN values, but none were found. Debugging by stepping through the code proved ineffective, as this issue can only be reliably analysed by examining a group of samples for each pixel.

Consequently, a decision was made to export all paths with relevant information for all samples within a small area, such as that shown in Figure 4.7. A Python script was developed to perform statistical analysis and help visualise ray positions, directions, and distributions.

We expected the edge to be well illuminated by indirect light, particularly by light reflecting off the short box. However, from the gathered information, we determined that hardly any light was being reflected towards the edge due to the narrow angle between the area light and the vertical face of the short box.

Anvil Approach

The first step was to identify which visualiser to use in this case. As the problem persisted across an area, it was logical to capture data across that area and visualise it. To achieve this, a web-based radiance and point visualiser was used. Furthermore, the ability to select points on the image and analyse the distribution of shadow and probe rays was required.

The second step involved understanding the composition of the visualiser's molecule. Once all the atoms were identified, the necessary mappings were implemented. These mappings enable Anvil to interpret the user's data structures. Finally, the renderer is executed and the required frame buffer is passed to Anvil at each frame.

Figure 4.8 illustrates a filtered view of a particular pixel within the web visualiser. This view shows all the points that the area light reached and was able to connect to the selected point on the problematic dark edge. It demonstrates that no light was being reflected from the short box to the tall box, and also that the short box acts as an occluder to indirect light from the surrounding environment.

Discussion

BPT was implemented on the CPU, and although debugging on the CPU is typically easier, the issue encountered could not be readily identified using conventional methods.

Both approaches share only the streaming of captured data in common. The conventional method streams data to a file, whereas the Anvil approach streams data directly to it. Visualising data using Python requires substantial effort. Furthermore, when plotting a large number of data points, interactivity suffers.

With Anvil, visualisation is provided automatically. Once the mappings are established, it is simply a matter of posting data to it. The web visualiser offers ready-made filters, and missing ones can be easily created by adding new predicates. Although the web visualiser is a separate process from Anvil, communication via sockets on the same machine makes it seamless. Additionally, Anvil enables the user to observe

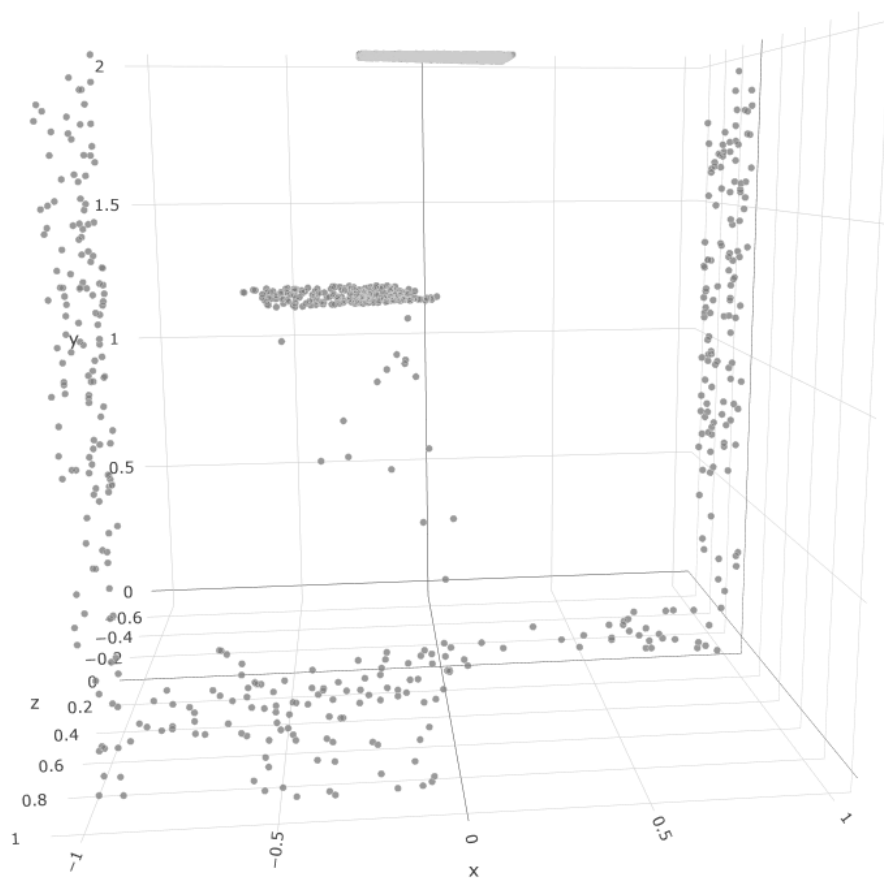


Figure 4.8 Web visualiser showing hit points at depth 2

points of interest appearing in real time as the renderer operates. This allows the user to pause rendering when a suspected precursor to a problem emerges and switch to conventional debugging if necessary.

Consequently, the effort required to debug this issue with Anvil is significantly less than that needed to export data from the renderer and import it into a Python application.

4.11.3 Verification Testing

In exploratory testing, an individual performs tests without following a predetermined script. Instead, the tester examines the system by questioning how it would react in specific scenarios. The results from previous tests can then guide further testing and exploration of other scenarios. One advantage over scripted testing is that bugs can be identified more rapidly, as no prior preparation is required.

In Section 4.11.1, a GPU path tracer with NEE is used. Although this path tracer appears to function correctly, it would be beneficial to verify whether certain properties, such as energy conservation, are maintained.

Traditional Approach

One of the simplest forms of testing is through assertions. These can be inserted at critical points in the algorithm to verify that the resulting calculations are correct. In this case study, an assertion was added to check that the energy contributed by all paths is positive. A second assertion tests whether light is being emitted from the back face of an area light. These assertions did not fail, which increased confidence in the implementation. However, test coverage remained low, and many other cases still required testing.

Anvil Approach

The approach taken using Anvil can be categorised as purely exploratory. The path visualiser was used to explore the scene while the renderer was running, and it was observed that a ray was being emitted from the back face of a primitive. This behaviour is undesirable as computation is wasted to generate a new ray and propagate it into the scene.

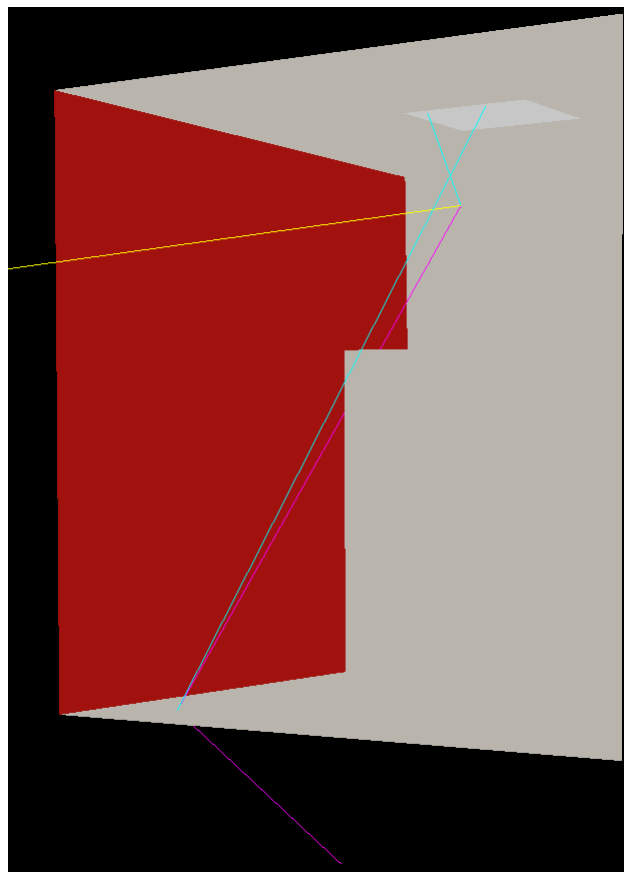


Figure 4.9 Anvil debug view during exploratory analysis

Figure 4.9 shows the offending ray at the very bottom. The ray information extracted from Anvil revealed that while the ray was facing in the correct direction, its

t value was negative. Consequently, the ray propagated backwards, which constitutes faulty behaviour.

Discussion

The first method relies solely on placing assertions, while the second approach involves linking Anvil to the application, mapping the path molecule, and streaming data to Anvil. Using assertions ensures that specific properties are consistently respected. Conversely, Anvil provides greater freedom to navigate the scene in a more natural way.

In this instance, Anvil proved valuable as this particular case had not been captured by the assertions. However, combining both testing methods increases confidence in the implementation and helps to minimise bugs to a certain extent.

4.12 Discussion

The evaluation in Section 4.11 suggests that Anvil does indeed assist in debugging graphics applications. It provides an interactive environment well suited for debugging and analysis. Anvil is straightforward to use and requires only a few steps to become operational. Once linked to an application, it only requires the user to implement the instrumentation steps discussed in Section 4.10, unless a suitable visualisation or analysis system is already available. The usability of Anvil depends on the number of systems provided out of the box, plugins shared by other users, and the ease of extensibility. The traditional approach may be simpler to set up initially, but Anvil provides a richer debugging experience.

Anvil is modular, and its systems can be extended easily through a simple interface. Furthermore, there is no mandatory direct dependency between different systems; they can communicate using a well-defined interface via the event manager. This aligns with the fundamental requirements for extensibility discussed in [18].

4.13 Summary

This chapter has presented Anvil, a visual debugging tool designed to address the complexities of PBR development. Our work began with an existing CPU-based path tracer in Candela, selected for its simplicity and modular design. However, initial testing revealed uncertainties in its correctness, necessitating the development of ad hoc diagnostic tools during the debugging process. Through iterative refinement, we recognised that these tools such as pixel radiance inspectors, path visualisers, and statistical analysers could be generalised into a unified framework. This realisation led to Anvil,

a system that not only resolved immediate debugging challenges but also provided a scalable foundation for future GPU adaptations.

The transition from problem-specific tools to Anvil was motivated by the inherent difficulties in debugging stochastic rendering algorithms, particularly on GPUs, where conventional debugging methods are inadequate. Anvil’s core design principles namely efficiency, extensibility, and flexibility were empirically validated through its application to three key tasks: debugging firefly artefacts in path tracing (Section 4.11.1), analysing energy distribution in bidirectional path tracing (Section 4.11.2), and exploratory testing of numerical stability (Section 4.11). These case studies demonstrated Anvil’s qualitative advantages over traditional approaches, such as NVIDIA Nsight or manual data export, by enabling real-time visualisation of high-level primitives (e.g., light paths and distributions) through its atom and molecule abstractions.

By the conclusion of this phase, we achieved two significant milestones: a correct CPU and GPU path tracer, and a correct bidirectional CPU path tracer. These implementations now serve as reference solutions for subsequent chapters, ensuring methodological rigour in our pursuit of real-time caustics rendering. Critically, Anvil’s utility extends beyond our immediate needs; its architecture is domain-agnostic, making it applicable to pedagogy (e.g., visualising light transport for educational purposes) and other fields requiring high-level debugging of spatial or stochastic processes.

5 Light Tracing

Caustics are intricate light patterns arising from the concentration of light via specular reflections or refractions onto diffuse surfaces. These patterns exhibit high-frequency signals, presenting a significant challenge for real-time rendering compared to the low-frequency signals typical of diffuse-dominated scenes. While path tracing (PT) is a versatile, unbiased technique capable of rendering all light phenomena given sufficient samples, its convergence for caustics is notoriously slow due to the low probability of capturing specular-to-diffuse light paths. This inefficiency motivates the exploration of alternative methods better suited to the task, such as light tracing (LT) and photon mapping (PM).

Among these alternatives, LT offers compelling advantages for GPU implementation. Like PT, LT is unbiased and stateless, requiring no complex data structures. This is a critical feature for exploiting modern ray-tracing hardware efficiently. In contrast, PM, despite its effectiveness for caustics, introduces bias, relies on multi-pass workflows, and demands costly spatial data structures (e.g., photon maps), complicating real-time adaptation. Manifold-based techniques, while unbiased and highly effective for caustics, are unsuitable for real-time due to their reliance on an iterative Newton solver, which is computationally expensive and divergent on a GPU. Furthermore, resampling-based techniques such as ReSTIR, whilst designed for real-time performance, struggle to render caustics effectively due to the challenges of reusing paths with high-frequency specular components. BPT, though unbiased and theoretically faster, suffers from computational overhead: connecting n eye paths with m light paths scales as $O(mn)$, making it less practical for initial prototyping. LT thus emerges as a pragmatic candidate, bridging simplicity and performance with its single-pass, ray-centric design.

To validate LT’s suitability for caustics, this chapter adopts a structured methodology. First, a CPU-based implementation, Candela Light Tracer (CLT), is developed. This serves multiple purposes: isolating algorithmic correctness from GPU-specific complexities, enabling thorough debugging via Anvil (a visual debugging framework introduced in Chapter 4), and establishing a ground-truth reference for subsequent GPU porting. Anvil’s role is pivotal here, allowing inspection of light paths and energy propagation at runtime, which is especially valuable for diagnosing sampling artefacts or numerical instabilities in stochastic methods.

The CPU implementation also facilitates quantitative evaluation. By comparing CLT against Candela’s verified PT implementation, we rigorously assess LT’s relative efficiency for caustics, quantifying speedups in convergence and highlighting scenarios where LT outperforms PT. Such analysis informs the GPU adaptation, where

LT's statelessness and ray coherence align ideally with hardware-accelerated ray tracing. The transition to GPU proceeds incrementally, with Anvil verifying each ported component (e.g., kernelised ray generation or importance sampling) against the CPU reference. This phased approach mitigates risks inherent in GPU debugging, where traditional tools often falter at reconstructing stochastic processes.

In summary, this chapter justifies LT as a middle ground between PT's generality and BPT's complexity, leveraging modern GPU capabilities while avoiding biases and overheads inherent in other algorithms such as those reviewed in Chapter 3. By combining a CPU reference implementation for verification, Anvil-driven debugging, and incremental GPU porting, we aim to demonstrate LT's viability for real-time caustics, laying groundwork for future hybrid techniques.

5.1 Light Tracing

A naive light tracer is one where paths start from the light source and are distributed equally and randomly in all directions. Consequently, no importance sampling occurs. Depending on the camera model, a light tracer collects either flux (for example, for a thin lens camera) or irradiance (for a pinhole camera) using the importance equation (Equation 2.51) coupled with the measurement equation (Equation 2.52). A final averaging step transforms flux or irradiance into radiance.

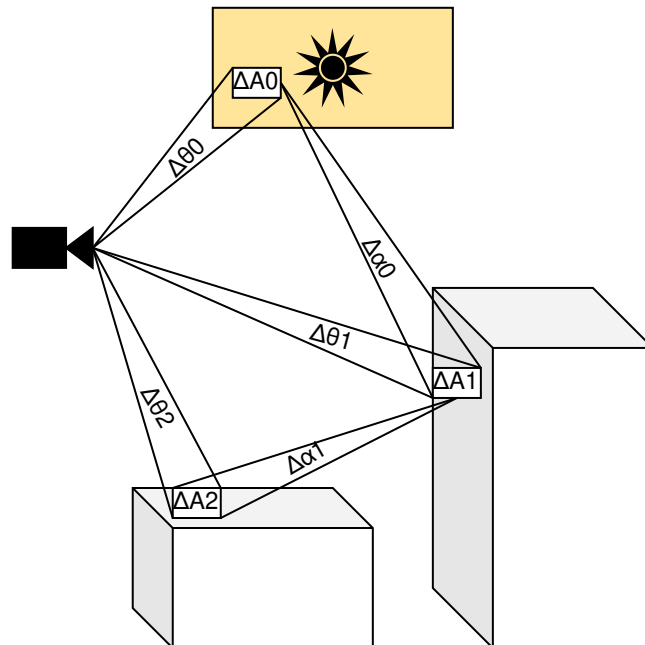


Figure 5.1 The process of light tracing with a pinhole camera

Figure 5.1 illustrates the process of light tracing when a pinhole camera model is used. A random point on the light source is selected, denoted as ΔA_0 . The total

irradiance flowing from ΔA_0 towards the camera within $\Delta\theta_0$ steradians is computed. A random direction subtending $\Delta\alpha_0$ steradians is then chosen, and the ray propagates until it intersects an object in the scene. In this instance, the ray strikes the tall box, forming a projected area ΔA_1 . Similarly, the irradiance flowing from ΔA_1 towards the camera within $\Delta\theta_1$ steradians is calculated. The cycle is repeated by selecting another random direction from ΔA_1 .

5.1.1 Derivation

The formulation for irradiance is described in Equation 5.1. The irradiance caused by direct light corresponds to the scenario where ΔA_0 emits radiance into the solid angle $\Delta\theta_0$. This solid angle $\Delta\theta_0$ is equal to $\frac{\cos(\theta_0)\Delta A_0}{r^2}$, using the substitution from surface area to solid angle (see Equation 2.35). Equation 5.1 is obtained by taking the limit as $\Delta A_0 \rightarrow 0$ and summing all the infinitesimal areas. Here, L_e denotes the emitted radiance, r is the distance between the camera and the point in question, $\cos(A)$ is the dot product of the normal of surface A with the direction vector towards the camera, and $\cos(\theta)$ represents the dot product of the camera's normal with the direction vector towards surface A .

$$E = \int_A L_e \cos(\theta) \frac{\cos(A)}{r^2} dA \quad (5.1)$$

The irradiance resulting from indirect light via the first bounce is calculated as follows. The total flux incident on ΔA_1 from ΔA_0 is $L_e \cos(\theta_0) \Delta\alpha_0 \Delta A_0$. The projected area of the box's surface in the direction of the light is $\Delta A_1 \cos(\theta_1)$, and the solid angle subtended by the light element with respect to the box's area element is $\Delta\Omega$. The average incident radiance on surface ΔA_1 is presented in Equation 5.2.

$$L_i = \frac{L_e \cos(\Delta A_0) \Delta\alpha_0 \Delta A_0}{\Delta A_1 \cos(\Delta A_1) \Delta\Omega} \quad (5.2)$$

The outgoing radiance from surface ΔA_1 to the camera is L_o . This value can be computed using the rendering equation.

$$L_o = \int_{\Omega} L_i \cos(\Delta A_1) f_r d\omega \quad (5.3)$$

Since radiance L_i is incident from a single direction with solid angle $\Delta\Omega$ in this instance, L_o evaluates to

$$L_o = L_i \cos(\Delta A_1) f_r \Delta\Omega = \left(\frac{L_e \cos(\Delta A_0) \Delta\alpha_0 \Delta A_0}{\Delta A_1 \cos(\Delta A_1) \Delta\Omega} \right) (f_r \cos(\Delta A_1) \Delta\Omega) \quad (5.4)$$

$$L_o = \frac{L_e \cos(\Delta A_0) \Delta \alpha_0 \Delta A_0}{\Delta A_1} f_r \quad (5.5)$$

Using the approximate surface formulation of the rendering equation as well as using the solid angle substitution ($\Delta \theta_1 = \frac{\cos(\Delta A_{1c}) \Delta A_1}{r^2}$, $\cos(\Delta A_{1c})$ is the dot product between the surface normal of ΔA_1 and the camera, and r is the distance between ΔA_1 and the camera), the whole expression becomes

$$\Delta E = \frac{L_o \cos \theta \Delta A_1 \cos(\Delta A_{1c})}{r^2} \quad (5.6)$$

Substituting L_o results in the following expression:

$$\Delta E = \frac{L_e \cos(\Delta A_0) \Delta \alpha_0 \Delta A_0 f_r}{\Delta A_1} \frac{\cos \theta \Delta A_1 \cos(\Delta A_{1c})}{r^2} \quad (5.7)$$

$$\Delta E = L_e \cos(\Delta A_0) f_r \cos \theta \frac{\cos(\Delta A_{1c})}{r^2} \Delta \alpha_0 \Delta A_0 \quad (5.8)$$

Taking the limit as $\Delta \alpha_0 \rightarrow 0$ and $\Delta A_0 \rightarrow 0$ and summing all the infinitesimal solid angles and surfaces, the incident irradiance becomes:

$$E = \int_A \int_\alpha L_e f_r \cos \theta \cos(A_0) \frac{\cos(A_{1c})}{r^2} d\alpha dA \quad (5.9)$$

The general recursive irradiance equation for a pinhole camera is:

$$E = \int_A W L_e dA \quad (5.10)$$

where

$$W = \cos \theta \frac{\cos A}{r^2} + \int_\Omega W f_r \cos \omega d\omega \quad (5.11)$$

Equation 5.10 enables the computation of irradiance at the camera's pinhole. To obtain radiance values for each pixel, this irradiance must be distributed across all pixels. Whenever an irradiance contribution is made, the corresponding pixel is identified and the irradiance is accumulated in that pixel's irradiance buffer. Equation 5.12 is used to calculate the average radiance across the solid angle subtended by a pixel P_Ω and is applied to every pixel to determine its radiance value. The term E_{P_Ω} represents the accumulated pixel irradiance. $P_r = \int_{P_\Omega} \cos \theta d\omega$ is a constant which differs per pixel and can be initialised when the program starts.

This value can be computed analytically, partly because there is no occlusion between the pinhole and the pixel plane. Another reason is that the pixel plane and the pinhole camera maintain a fixed relative orientation; the pixel plane's normal and the pinhole camera's normal are always parallel. These simplifications enable calcula-

tions using traditional calculus. Equation 5.13 presents the antiderivative for this integral, and Equation 5.14 shows the antiderivative with the integral bounds incorporated. Here, x_0 , y_0 , x_1 , and y_1 are the world space coordinates defining the pixel square, and z is the distance from the pinhole to the pixel plane.

$$L_{Avg} = \frac{\int_{P_\Omega} L \cos \theta d\omega}{\int_{P_\Omega} \cos \theta d\omega} = \frac{E_{P_\Omega}}{P_r} \quad (5.12)$$

$$P_t(x, y, z) = \int \cos \theta d\omega = \frac{1}{2} \left(\frac{x \arctan \left(\frac{y}{\sqrt{x^2 + z^2}} \right)}{\sqrt{x^2 + z^2}} + \frac{y \arctan \left(\frac{x}{\sqrt{y^2 + z^2}} \right)}{\sqrt{y^2 + z^2}} \right) \quad (5.13)$$

$$P_r = P_t(x_1, y_1, z) + P_t(x_0, y_0, z) - P_t(x_1, y_0, z) - P_t(x_0, y_1, z) \quad (5.14)$$

5.2 Candela

Candela is a CPU-based renderer featuring a path tracer implementation. This section first outlines the materials supported by Candela and describes its path tracing algorithm, then details the implementation of our light tracer, referred to as CLT. We subsequently compare the performance of PT and CLT across a range of scenes to evaluate their efficacy in rendering caustics. This comparison serves to identify the most suitable candidate for GPU implementation.

5.2.1 Materials

The materials in Candela utilise a smooth compound BSDF model made up of three components: a Lambertian BRDF for diffuse reflection, a specular BRDF for perfect mirror reflection, and a specular BTDF for perfect transmission. The model is governed by three material coefficients k_d , k_s and k_t representing diffuse, specular, and transmissive components respectively which are constrained by energy conservation such that $k_d + k_s + k_t = 1$. When a ray intersects a material surface, the BSDF sampling procedure stochastically selects one component with probability proportional to its coefficient. For the Lambertian BRDF, cosine-weighted hemisphere sampling is used, which importance-samples directions according to the projected solid angle distribution characteristic of Lambertian materials. For reflection, the ray is perfectly reflected without attenuation, while for transmission, the ray refracts according to Snell's law based on the material's refractive index. Volumetric absorption through participating media is simulated using Beer's law, accounting for material thickness and absorption proper-

ties, while the coefficient normalisation ensures the BSDF maintains energy conservation throughout the light transport process.

Figure 5.2 illustrates how an incoming ray (shown in red) may be split according to the Candela material model. The red ray strikes the material, with a proportion k_d being diffusely reflected (indicated by green arrows), k_r being reflected (top blue arrow), and k_t being transmitted (top orange arrow). At the subsequent intersection, which occurs within the material (after absorption governed by Beer’s law, e^{-sk_b} , where k_b is the absorption coefficient and s is the distance travelled through the material), the ray may undergo total internal reflection depending on the angle of incidence, or it may be transmitted, exiting the material. Both outcomes are subject to a throughput factor of $k_t^2 e^{-dk_b}$.

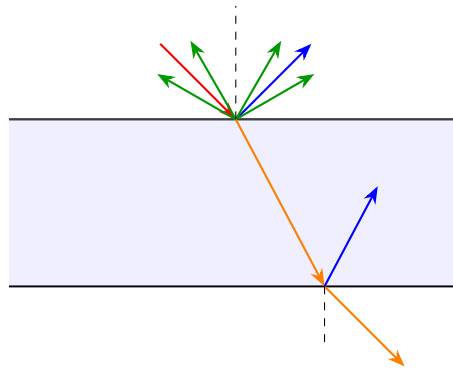


Figure 5.2 Candela Material. Rays: red is incoming, green is diffuse, blue is reflection and orange is refraction.

5.2.2 Candela Path Tracer

The Candela Path Tracer implements a forward path tracing algorithm (depicted in Algorithm 1, with variable definitions in Table 5.1) that constructs paths from the camera into the scene, accumulating radiance contributions through multiple light bounces. The algorithm begins by initialising a radiance accumulator and path throughput variable, then iteratively traces rays through the scene geometry. At each intersection point, the algorithm handles participating media through Beer’s law attenuation, applying exponential decay based on the material’s absorption coefficient and travel distance.

The core of the algorithm employs NEE for direct lighting calculation, uniformly sampling light sources and testing visibility through shadow rays, while computing the geometric term G to account for the cosine-weighted solid angle relationship. Path continuation uses BSDF importance sampling to determine subsequent directions, updating the path throughput T with the ratio of the BSDF evaluation to the sampling probability. The algorithm incorporates Russian roulette termination after three bounces,

maintaining unbiased results through probabilistic path termination with throughput compensation.

The rendering algorithm relies on NEE to account for light emission. However, NEE fails with specular materials, as their perfectly deterministic bounce direction (dictated by the incident direction and material properties) prevents the construction of a connection path to a light source. To solve this, the system directly accumulates emissive light after a specular bounce, if an emissive material is hit. This exception handling guarantees that mirrors, glass, and other specular surfaces are lit correctly. However, this works only with area-based lights because specular surfaces cannot connect to point or point-based lights such as spot lights. Consequently, path tracing fails to capture the caustics these lights produce.

Variable	Definition
$A(P)$	Surface area of light source P
$G(x \leftrightarrow x_{\text{light}})$	Geometric term between surfaces
L	Accumulated radiance along the path
L_{direct}	Direct lighting contribution
$L_e(x_{\text{light}})$	Emitted radiance at light point x_{light}
$\mathcal{L}$	Set of light sources (emissive primitives)
P	Sampled light source
$\mathcal{S}$	Scene geometry and materials
T	Path throughput (weight of current path segment)
$\mathcal{U}$	Random sampler (pseudo-random number generator)
f	BSDF: Compounds BRDF and BTDF
f_r	BRDF
i	Bounce counter (number of surface interactions)
$\mathbf{m}$	Material properties at intersection
$\mathbf{n}$	Surface normal at intersection
$\mathbf{n}_{\text{light}}$	Normal at sampled light point
p_{cont}	Russian roulette continuation probability
p_{ω}	PDF of the sampled direction ω_o
$\mathbf{r}$	Current ray being traced
$\mathbf{r}_{\text{in}}$	Input ray (origin + direction)
$\mathbf{r}_{\text{shadow}}$	Shadow ray for visibility testing
t	Ray intersection distance
x	Intersection point
x_{light}	Sampled point on light source
δ_{prev}	Flag indicating previous bounce was Dirac delta (specular/glass)
ϵ	Small offset to avoid self-intersection
θ_{light}	Angle between shadow ray and light normal
θ_{surf}	Angle between shadow ray and surface normal
σ_a	Absorption coefficient for participating media
$\omega, \omega_{\text{light}}$	Emission direction from light source
ω_o	Sampled outgoing direction for next bounce

Table 5.1 Variable Definitions for all Algorithms

5.2.3 Candela Light Tracer

The Candela Light Tracing algorithm depicted in Algorithm 2 (with variable definitions in Table 5.1 and Table 5.2) implements a pure backward path tracing approach that constructs light paths from emissive sources. The algorithm begins by uniformly selecting a light source from the scene’s emissive primitives ($\mathcal{L}$) and sampling a point on its surface. For area lights, this typically involves cosine-weighted sampling to sample directions proportionally to the cosine between the surface normal and outgoing direction. The initial path throughput T accounts for the light’s emitted radiance L_e , surface area $A(P)$, and selection probability.

For subsequent bounces, the sampling strategy depends on the material properties at each intersection point. Lambertian materials use cosine-weighted sampling, while specular materials use perfect reflection and transmission directions, and complex BSDFs use their own importance sampling strategies tailored to their scattering characteristics. At each path vertex, the algorithm checks for direct connections to the camera, evaluating the geometric term G that encodes the cosine-weighted solid angle relationship between the light point and camera, and accumulating irradiance contributions to corresponding pixels when paths are unoccluded.

The path throughput accumulates the product of BSDF evaluations, cosine terms, and probability density function reciprocals, maintaining the path’s weight while accounting for participating media attenuation via Beer’s law for internal intersections. Russian roulette termination after three bounces ensures unbiased rendering with finite computational resources by probabilistically terminating paths while preserving energy through throughput compensation. The algorithm ultimately accumulates irradiance (E_{buffer}) at each pixel arriving at the sensor from all contributing light paths.

Light tracing shares a key limitation with path tracing where it cannot handle certain specular surface events. Just as path tracing struggles to sample emissive objects through a specular bounce, and cannot connect to point-based lights, light tracing cannot connect light paths from specular surfaces to the pinhole camera, causing them to render as black.

5.2.4 Methodology

The evaluation of the CLT employs a systematic approach to assess its effectiveness in rendering caustics compared to traditional PT. A selection of carefully designed test scenes is used, ranging from purely diffuse environments to scenes combining diffuse and specular materials. These scenes are chosen to isolate the performance of each algorithm under different lighting and material conditions, with particular attention given to scenarios where caustics dominate the illumination.

Algorithm 1 Candela Path Tracer

```

1: function PATHTRACE( $\mathcal{S}, \mathbf{r}_{\text{in}}, \mathcal{U}$ )
2:   Input: Scene  $\mathcal{S}$ , Ray  $\mathbf{r}_{\text{in}}$ , Sampler  $\mathcal{U}$ 
3:   Output: Radiance  $L$ 
4:   if  $\mathcal{L} = \emptyset$  then return ▷  $\mathcal{L}$ : light sources (emissive primitives)
5:   end if
6:    $L \leftarrow 0, T \leftarrow 1$  ▷ Initialise radiance and throughput
7:    $\mathbf{r} \leftarrow \mathbf{r}_{\text{in}}$ 
8:    $i \leftarrow 0$  ▷ Bounce counter
9:    $\delta_{\text{prev}} \leftarrow \text{true}$  ▷ Previous bounce was Dirac delta
10:  while true do
11:     $(x, \mathbf{n}, \mathbf{m}, t) \leftarrow \text{Intersect}(\mathbf{r}, \mathcal{S})$  ▷ Returns intersection distance  $t$ 
12:    if no intersection then break
13:    end if
14:    if internal intersection then ▷ Handle participating media (Beer's law)
15:       $\sigma_a \leftarrow \mathbf{m}.\text{absorptionCoefficient}$ 
16:       $T \leftarrow T \cdot \exp(-\sigma_a \cdot t)$  ▷ Use distance  $t$  traveled
17:    end if
18:    if  $\delta_{\text{prev}} \wedge \neg \text{internal} \wedge \mathbf{m}.\text{isEmissive}()$  then ▷ Handle emissive surfaces
19:       $L \leftarrow L + T \cdot \mathbf{m}.\text{getEmissive}()$ 
20:    end if
21:     $P \sim \mathcal{U}(\mathcal{L})$  ▷ NEE Phase - Uniform light selection
22:     $(x_{\text{light}}, \mathbf{n}_{\text{light}}) \leftarrow \text{SampleSurface}(P, \mathcal{U})$ 
23:     $\mathbf{r}_{\text{shadow}} \leftarrow \text{Ray}(x, x_{\text{light}} - x, \epsilon)$ 
24:    if  $\text{Unoccluded}(\mathbf{r}_{\text{shadow}}, P) \wedge \cos \theta_{\text{surf}} > 0 \wedge \cos \theta_{\text{light}} > 0$  then
25:       $G(x \leftrightarrow x_{\text{light}}) \leftarrow \frac{\cos \theta_{\text{surf}} \cos \theta_{\text{light}}}{\|x - x_{\text{light}}\|^2}$ 
26:       $L_{\text{direct}} \leftarrow |\mathcal{L}| \cdot A(P) \cdot L_e(x_{\text{light}}) \cdot G(x \leftrightarrow x_{\text{light}})$ 
27:       $f_r \leftarrow \text{BRDF}(\mathbf{m}, -\mathbf{r}.\text{direction}, \omega_{\text{light}})$ 
28:       $L \leftarrow L + T \cdot L_{\text{direct}} \cdot f_r$ 
29:    end if
30:    if  $i \geq 3$  then ▷ Russian roulette termination
31:       $p_{\text{cont}} \leftarrow 0.5$  ▷ Survival probability
32:      if  $\mathcal{U}() > p_{\text{cont}}$  then break
33:    end if
34:     $T \leftarrow T / p_{\text{cont}}$  ▷ Adjust throughput for survived path
35:    end if
36:     $(\omega_o, f, p_\omega) \leftarrow \text{SampleBSDF}(\mathbf{m}, -\mathbf{r}.\text{direction}, \mathbf{n}, \mathcal{U})$  ▷ Sample direction
37:     $T \leftarrow T \cdot \frac{f \cdot |\omega_o \cdot \mathbf{n}|}{p_\omega}$  ▷ Update throughput
38:     $\mathbf{r} \leftarrow \text{Ray}(x, \omega_o, \epsilon)$ 
39:     $\delta_{\text{prev}} \leftarrow \text{IsDiracDelta}(\mathbf{m}, \omega_o)$ 
40:     $i \leftarrow i + 1$ 
41:  end while
42:  return  $L$ 
43: end function

```

Algorithm 2 Candela Light Tracer

```

1: procedure LIGHTTRACE( $\mathcal{S}, E_{\text{buffer}}, \mathcal{U}$ )
2:   Input: Scene  $\mathcal{S}$ , Irradiance buffer  $E_{\text{buffer}}$ , Uniform sampler  $\mathcal{U}$ 
3:   Output: Updated irradiance buffer  $E_{\text{buffer}}$ 
4:   if  $\mathcal{L} = \emptyset$  then return
5:   end if ▷  $\mathcal{L}$ : light sources (emissive primitives)
6:    $P \sim \mathcal{U}(\mathcal{L})$  ▷ Select light source uniformly
7:    $(x_{\text{light}}, \mathbf{n}_{\text{light}}) \leftarrow \text{SampleSurface}(P, \mathcal{U})$  ▷ Light  $P$  Surface sampling
8:    $T \leftarrow |\mathcal{L}| \cdot A(P) \cdot L_e(x_{\text{light}})$  ▷ Path throughput
9:    $\mathbf{r}_{\text{cam}} \leftarrow \text{Ray}(x_{\text{light}}, x_{\text{cam}} - x_{\text{light}}, \epsilon)$  ▷ Light to camera
10:  if Unoccluded( $\mathbf{r}_{\text{cam}}$ )  $\wedge$  GetPixel( $\mathbf{r}_{\text{cam}}, p$ ) then
11:     $G(x_{\text{light}} \leftrightarrow x_{\text{cam}}) \leftarrow \frac{\cos \theta_{\text{light}} \cos \theta_{\text{cam}}}{\|x_{\text{light}} - x_{\text{cam}}\|^2}$ 
12:     $E_{\text{buffer}}[p] \leftarrow E_{\text{buffer}}[p] + T \cdot G(x_{\text{light}} \leftrightarrow x_{\text{cam}})$ 
13:  end if
14:   $(\omega, p_{\omega}) \leftarrow \text{SampleLightDirection}(P, x_{\text{light}}, \mathbf{n}_{\text{light}}, \mathcal{U})$ 
15:   $T \leftarrow T \cdot \frac{|\omega \cdot \mathbf{n}_{\text{light}}|}{p_{\omega}}$  ▷ Update throughput with emission sampling
16:   $i \leftarrow 1$  ▷ Bounce count
17:   $\mathbf{r} \leftarrow \text{Ray}(x_{\text{light}}, \omega, \epsilon)$ 
18:  while true do
19:     $(x, \mathbf{n}, \mathbf{m}, t) \leftarrow \text{Intersect}(\mathbf{r}, \mathcal{S})$  ▷ Returns intersection distance  $t$ 
20:    if no intersection then break
21:    end if
22:    if internal intersection then ▷ Handle participating media
23:       $\sigma_a \leftarrow \mathbf{m}.\text{absorptionCoefficient}$ 
24:       $T \leftarrow T \cdot \exp(-\sigma_a \cdot t)$  ▷ Use distance  $t$  traveled
25:    end if
26:     $\mathbf{r}_{\text{cam}} \leftarrow \text{Ray}(x, x_{\text{cam}} - x, \epsilon)$  ▷ Surface to camera
27:    if Unoccluded( $\mathbf{r}_{\text{cam}}$ )  $\wedge$  GetPixel( $\mathbf{r}_{\text{cam}}, p$ ) then
28:       $f_r \leftarrow \text{BRDF}(\mathbf{m}, -\mathbf{r}.\text{direction}, \omega_{\text{cam}})$ 
29:       $G(x \leftrightarrow x_{\text{cam}}) \leftarrow \frac{\cos \theta \cos \theta_{\text{cam}}}{\|x - x_{\text{cam}}\|^2}$ 
30:       $E_{\text{buffer}}[p] \leftarrow E_{\text{buffer}}[p] + T \cdot f_r \cdot G(x \leftrightarrow x_{\text{cam}})$ 
31:    end if
32:    if  $i \geq 3$  then ▷ Russian roulette termination
33:       $p_{\text{cont}} \leftarrow 0.5$  ▷ Fixed survival probability
34:      if  $\mathcal{U}() > p_{\text{cont}}$  then break
35:      end if
36:       $T \leftarrow T / p_{\text{cont}}$  ▷ Adjust throughput for survived path
37:    end if
38:     $(\omega_{\text{next}}, f, p_{\omega}) \leftarrow \text{SampleBSDF}(\mathbf{m}, -\mathbf{r}.\text{direction}, \mathbf{n}, \mathcal{U})$  ▷ Next direction
39:     $T \leftarrow T \cdot \frac{f \cdot |\omega_{\text{next}} \cdot \mathbf{n}|}{p_{\omega}}$  ▷ Throughput update
40:     $\mathbf{r} \leftarrow \text{Ray}(x, \omega_{\text{next}}, \epsilon)$ 
41:     $i \leftarrow i + 1$ 
42:  end while
43: end procedure

```

Variable	Definition
E_{buffer}	Irradiance buffer storing accumulated illumination per pixel
$\mathbf{p}$	Pixel coordinates for camera projection
$\mathbf{r}_{\text{cam}}$	Ray connecting light/surface point to camera
x_{cam}	Camera position/origin
θ	Angle between $\mathbf{r}_{\text{cam}}$ and surface normal
θ_{cam}	Angle between $\mathbf{r}_{\text{cam}}$ and camera normal
θ_{light}	Angle between $\mathbf{r}_{\text{cam}}$ and light normal
ω_{cam}	Direction from surface point to camera
ω_{next}	Next sampled direction in light path

Table 5.2 Variable Definitions Specific to Algorithm 2

To quantify the performance of CLT, SSIM is employed as a primary metric. SSIM provides a perceptually meaningful measure of image quality by comparing luminance, contrast, and structure between the rendered output and a high-quality reference image. This allows for an objective assessment of how well CLT preserves fine details in caustic patterns, particularly in high-frequency regions where traditional methods often struggle.

A direct comparison between CLT and PT is conducted by rendering identical scenes with both methods using matched sample counts. The results are analysed not only in terms of global image quality but also through localised examinations of regions where caustics are the dominant light transport phenomenon. This targeted comparison helps isolate the strengths and weaknesses of each technique, revealing whether CLT’s light-first approach indeed offers superior convergence for caustics.

The analysis focuses on identifying the strengths and limitations of CLT in contrast to PT. By examining areas where caustics contribute significantly to the final image, the methodology highlights the conditions under which light tracing provides a measurable advantage. This comparative framework not only validates the efficiency of CLT for caustic rendering but also establishes a foundation for future optimisations and hybrid rendering approaches.

5.2.5 Evaluation

CLT was evaluated on various scenes consisting of objects composed of different materials. First, purely diffuse scenes were evaluated, and then specular materials were introduced. Part of the evaluation involved verifying the correctness of CLT by comparing its output with that of a path tracer.

Cornell Box

The Cornell box is a scene composed entirely of diffuse materials. It consists of a room containing an area light at the very top and two boxes, one tall and one short. Additionally, the left and right walls are coloured to enable the observation of light bleeding effects.

Figure 5.3 shows the Cornell box scene rendered with a path tracer on the left and with CLT on the right. Both images were rendered using 737,280,000 samples, equivalent to 2048 spp given the image resolution of 600 by 600 pixels. The CLT image achieves an SSIM score of 0.98 relative to the path-traced reference. This indicates a close match and confirms that CLT converges to the same result. Furthermore, an external web-based tool forming part of Anvil was used to compare radiance values at the pixel level (see Figure 5.4). Figures 5.5 and 5.6 illustrate the convergence of the images with increasing sample counts of 1, 4, 16, 64, and 256 spp. At each stage, the sample count is increased by a factor of four.

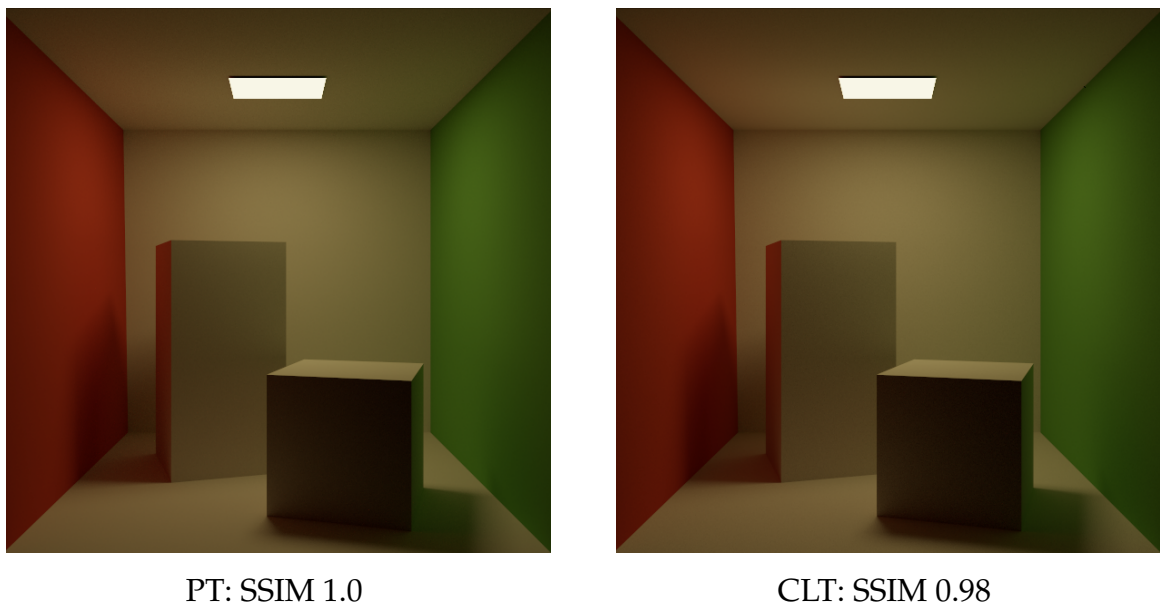


Figure 5.3 The Cornell box rendered with PT and CLT at 2048 spp

Based on the results from these images, it is observed that for purely diffuse scenes, path tracing is superior. The reason is that in path tracing, next event estimation is used at every intersection, thereby explicitly sampling a light source. In contrast, with CLT, samples are distributed according to the power distribution of the light source. Light paths can be lost if they do not pass through any pixel. The extent to which this occurs depends primarily on the scene type and the sampling strategies employed.

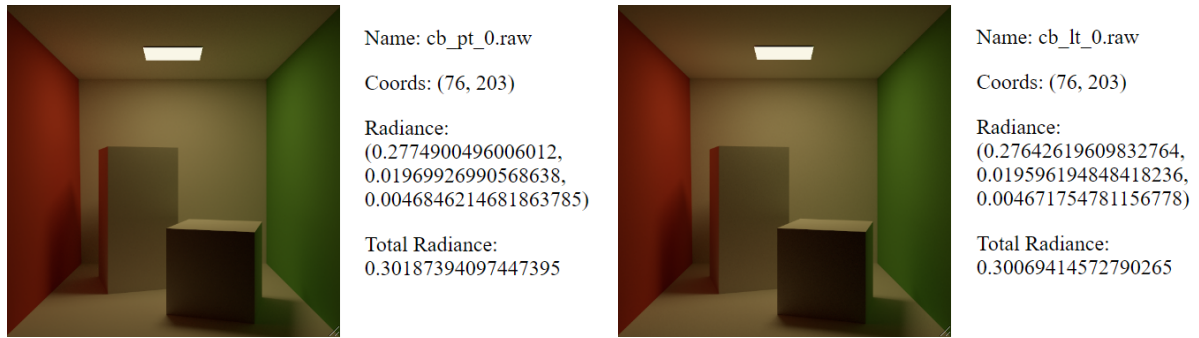
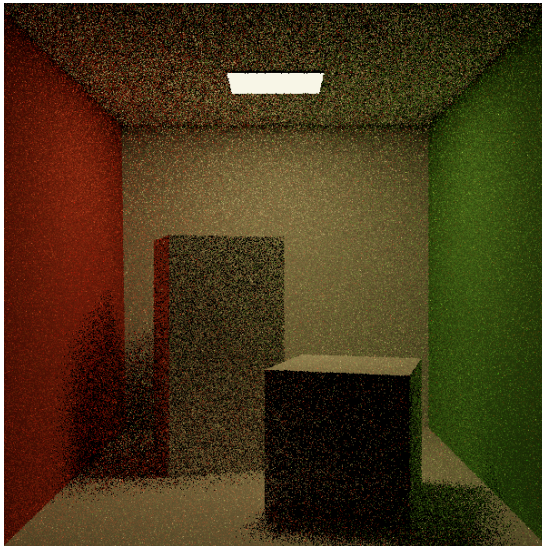


Figure 5.4 Anvil web radiance viewer

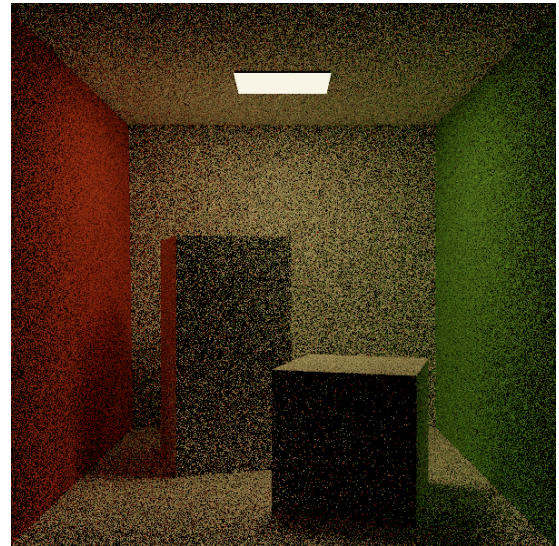
Glass Sphere

The glass sphere scene is a modified version of the Cornell box. The boxes are replaced by a hovering glass sphere. The sphere is not centred directly under the area light, but is positioned closer to the viewer. The optical properties of the sphere bend the light, concentrating it on the floor.

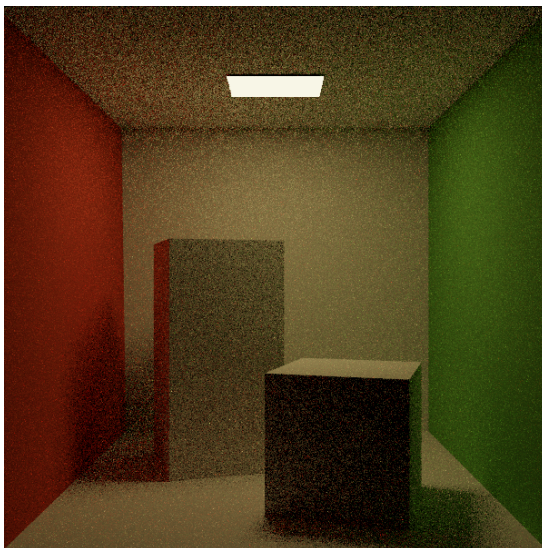
Figures 5.7 and 5.8 illustrate the performance of the path tracing algorithm and CLT at varying sample sizes. It is important to note that the sphere appears black in the CLT render because light rays cannot connect back to a pinhole camera when using light tracing. The probability of sampling a point in space is zero. CLT performs impressively in this scenario. The caustics appear immediately, even with just one samples per pixel. The caustics rendered by the path tracer at 256 spp are inferior to those produced by CLT at 16 samples per pixel. Figure 5.9 compares the results against a light-traced ground truth.



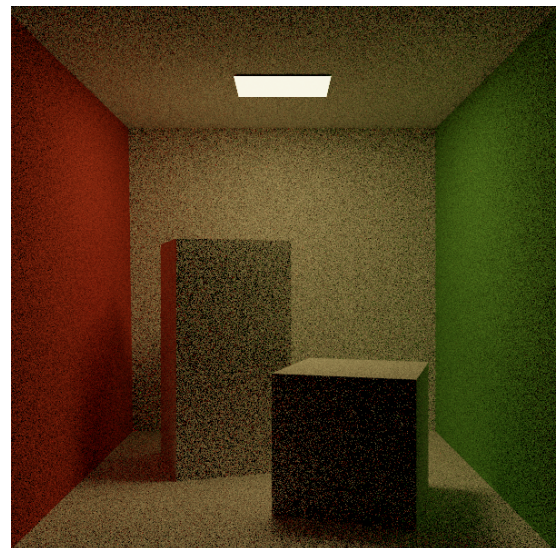
PT: 1 spp, SSIM 0.21



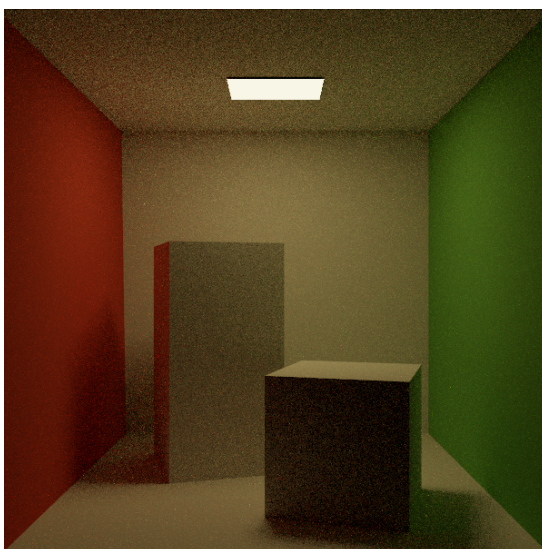
CLT: 1 spp, SSIM 0.1



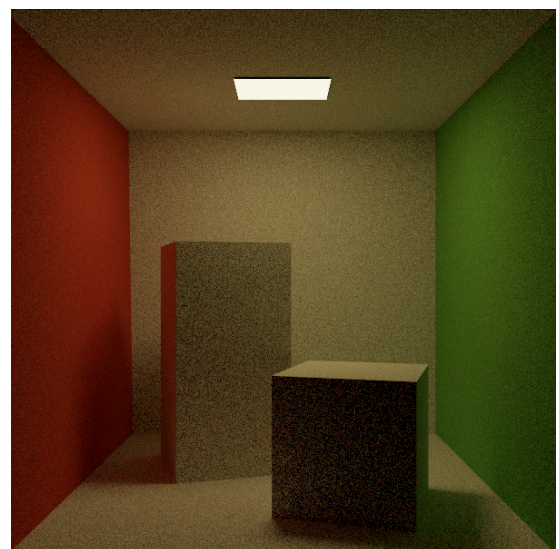
PT: 4 spp, SSIM 0.37



CLT: 4 spp, SSIM 0.27

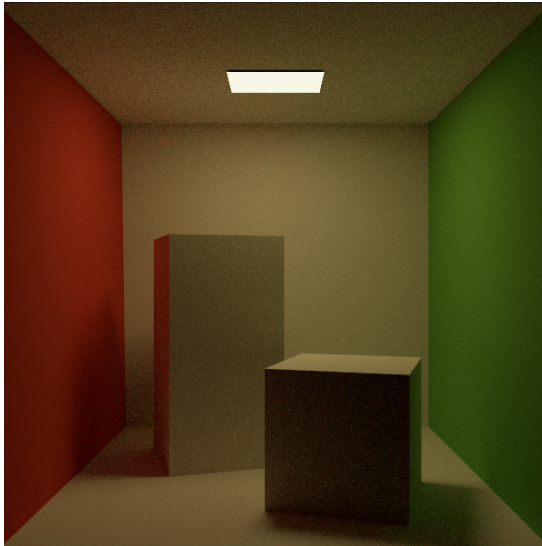


PT: 16 spp, SSIM 0.61

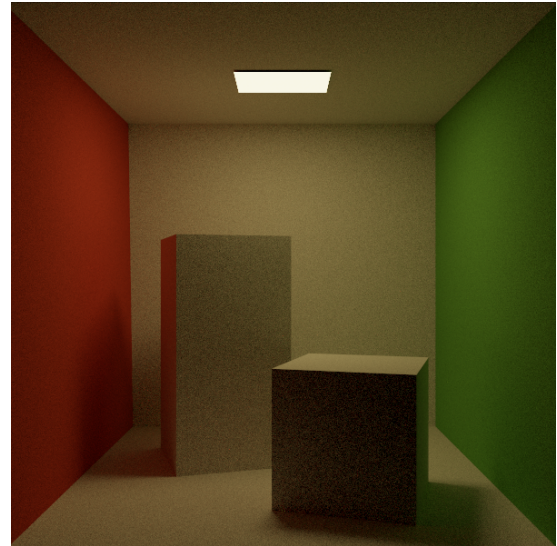


CLT: 16 spp, SSIM 0.55

Figure 5.5 The Cornell box rendered with PT and CLT at 1, 4, and 16 spp.



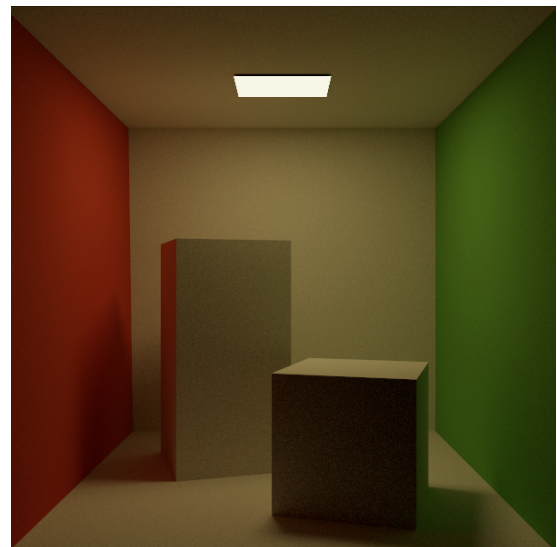
PT: 64 spp, SSIM 0.81



CLT: 64 spp, SSIM 0.79

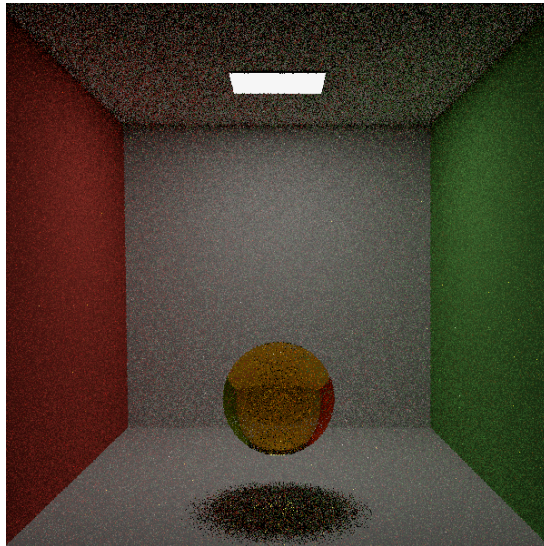


PT: 256 spp, SSIM 0.92

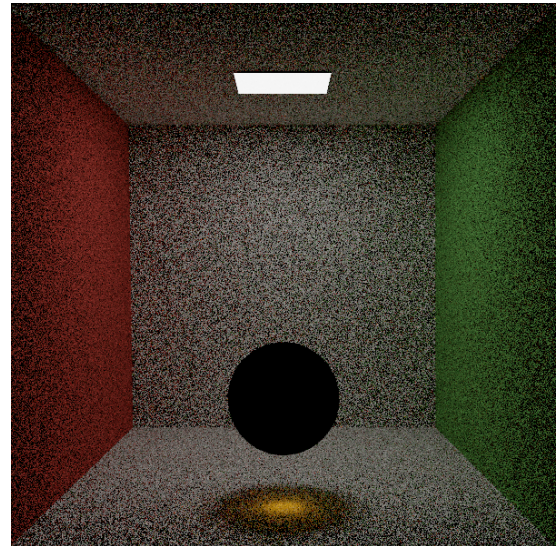


CLT: 256 spp, SSIM 0.92

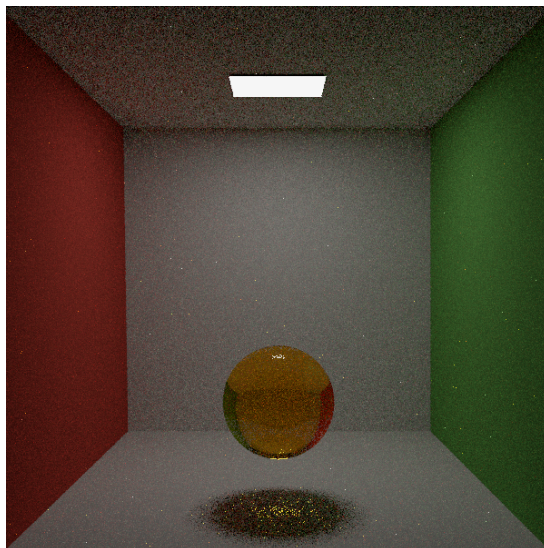
Figure 5.6 The Cornell box rendered with PT and CLT at 64 and 256 spp.



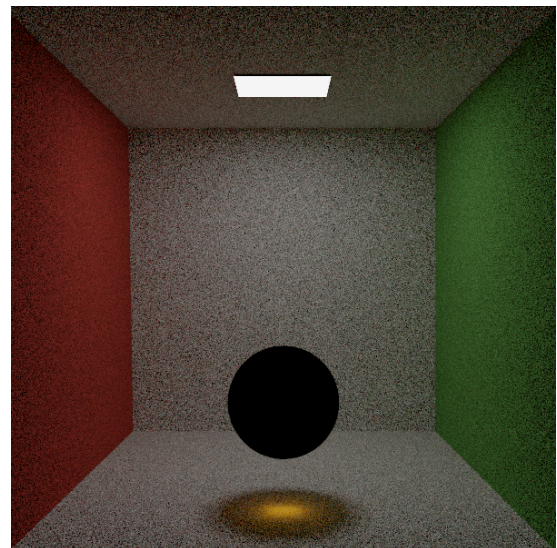
PT: 1 spp



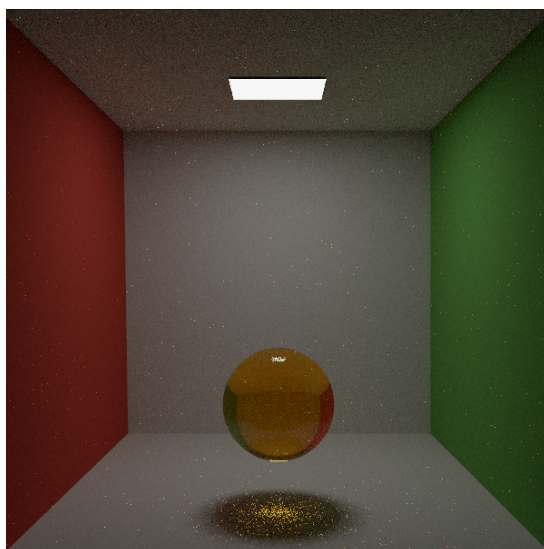
CLT: 1 spp



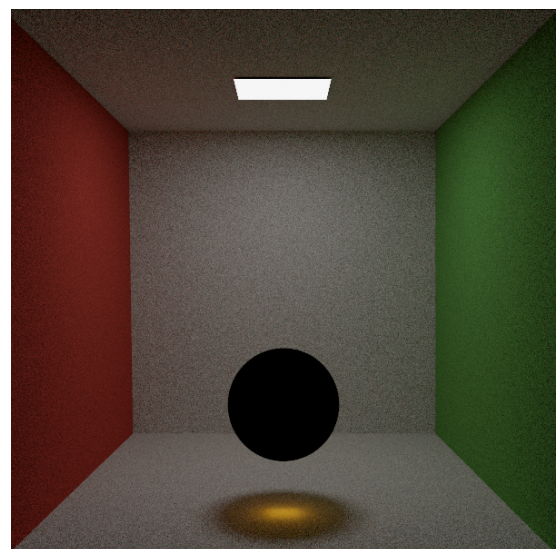
PT: 4 spp



CLT: 4 spp

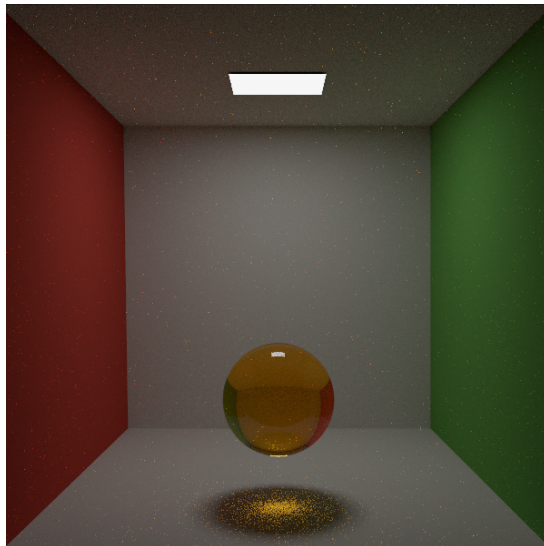


PT: 16 spp

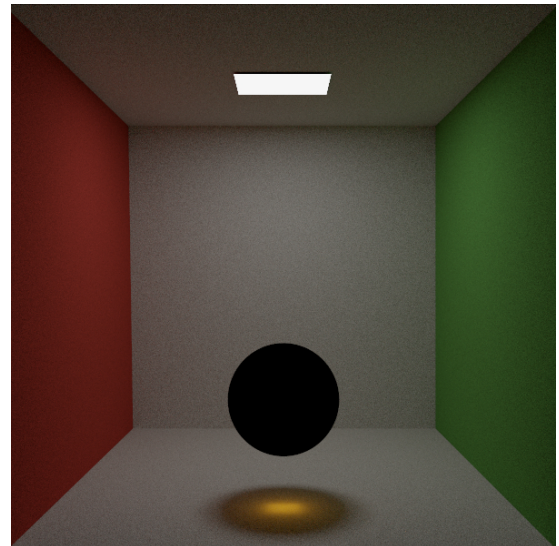


CLT: 16 spp

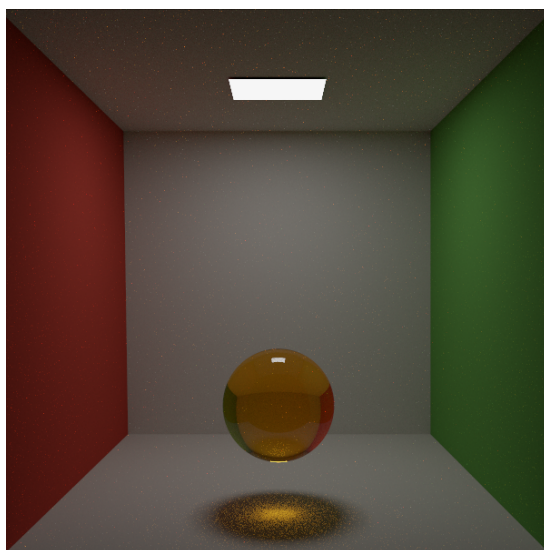
Figure 5.7 The glass sphere rendered with PT and CLT at 1, 4, and 16 spp



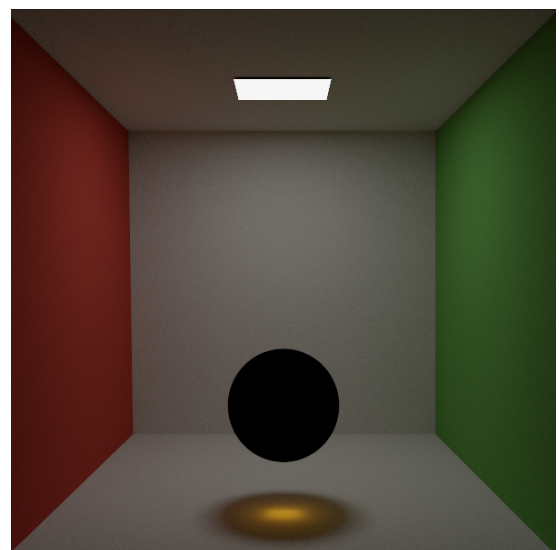
PT: 64 spp



CLT: 64 spp



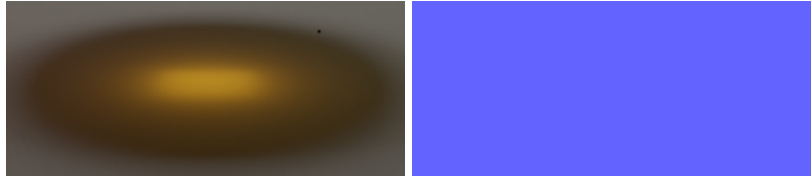
PT: 256 spp



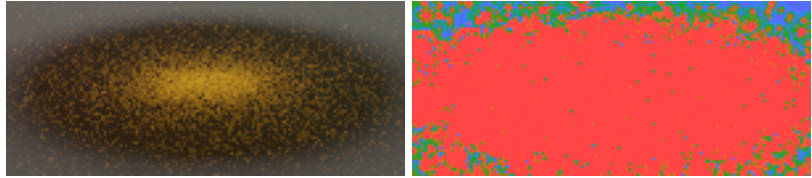
CLT: 256 spp

Figure 5.8 The glass sphere rendered with PT and CLT at 64 and 256 spp

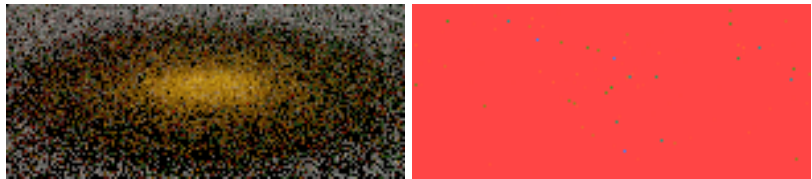
CLT (16384 samples)
Ground truth
Q = 100.00



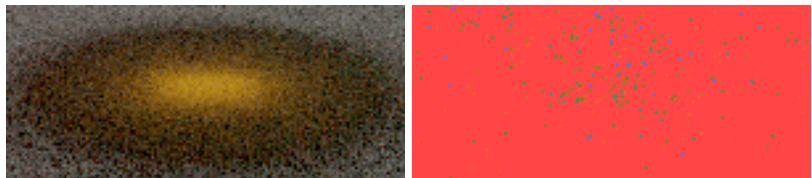
PT (256 spp)
Q = 47.72



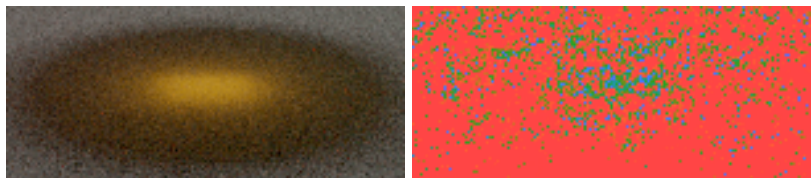
CLT (1 sample)
Q = 41.01



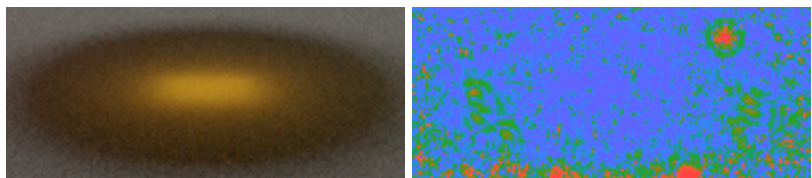
CLT (4 samples)
Q = 45.64



CLT (16 samples)
Q = 49.86



CLT (64 samples)
Q = 55.02



CLT (256 samples)
Q = 59.04

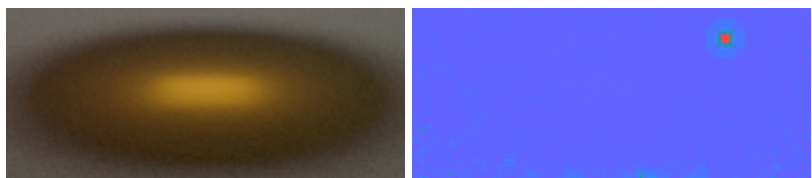


Figure 5.9 Comparison of a detail from Figures 5.7 and 5.8 against ground truth using HDRVDP version 2.2.2. Q is a metric returned by HDRVDP; a value of 100 represents the best quality (perfect match) and lower values represent lower quality. The path-traced image (caustic) at 256 spp is of lower quality than light-traced (CLT) images at 16 samples.

Ring

The ring scene is a modified version of the Cornell box, where the two boxes are replaced by a ring. The light source has been relocated downwards to the far back of the Cornell box, as shown in Figure 5.10. It is also oriented facing downwards, which presents a challenge for CLT. The ring is composed of a mirror-like reflective material;



Figure 5.10 The ring scene - observation of setup

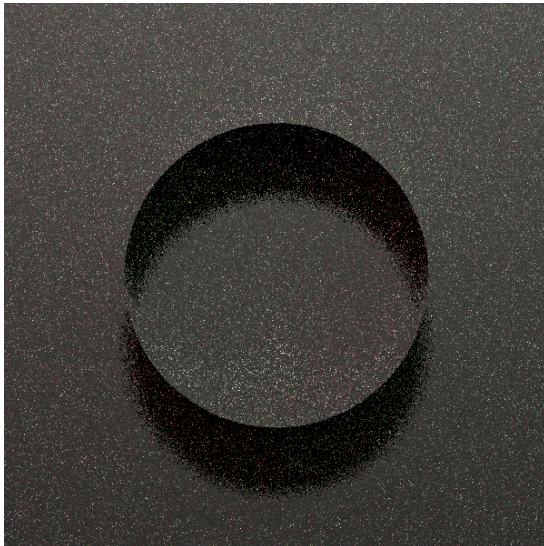
it is not metallic and reflects 90% of the light while diffusing the remainder. The angle between the light source and the ring is sufficiently steep to form a cardioid pattern on the floor. The camera is positioned directly above the ring, looking down upon it.

Figures 5.11 and 5.12 show the path tracing algorithm and CLT at varying sample sizes. The fundamental nature of each algorithm is immediately apparent at one samples per pixel, as observed through the image brightness. The next event estimation employed in path tracing ensures that pixels depicting diffuse surfaces receive light. In contrast, with CLT, light reaches the observed area far less frequently in this configuration. This also indicates that many samples in CLT are wasted, as the light rays seldom connect to the pinhole camera. It is important to note that the circumference of the ring appears bolder in the CLT render because light tracing does not connect specular rays with a pinhole camera.

The cardioid pattern that emerges with an increasing number of samples is essentially equivalent for both algorithms. CLT is expected to be superior for rendering caustics. However, in this scene, only a few light rays reach the relevant area, and caustics require light rays in order to form.

5.2.6 Discussion

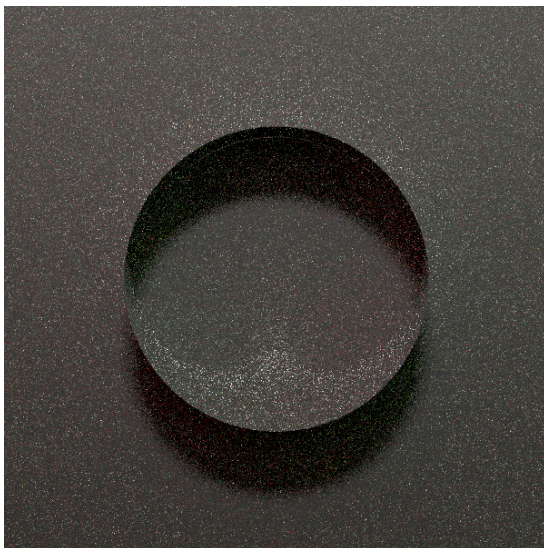
The evaluation results presented in Section 5.2.5 demonstrate light tracing's particular effectiveness for rendering caustic effects. The CLT achieved exceptional results for re-



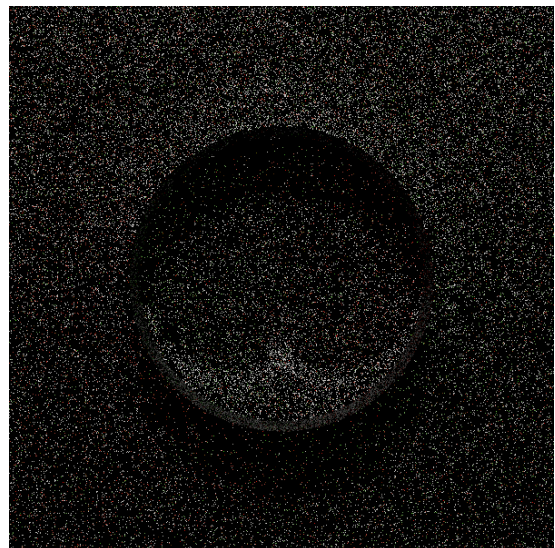
PT: 1 spp



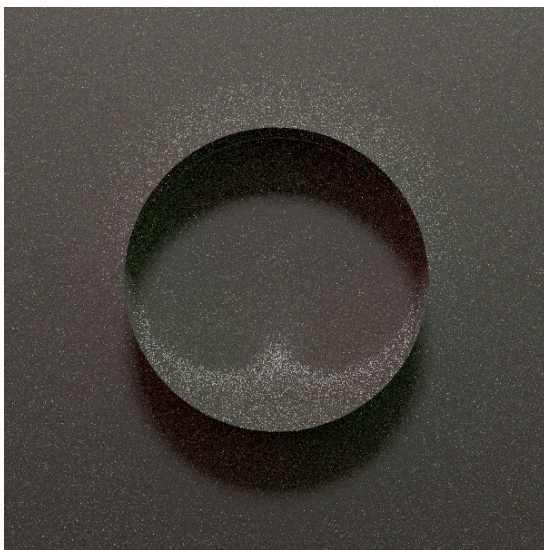
CLT: 1 spp



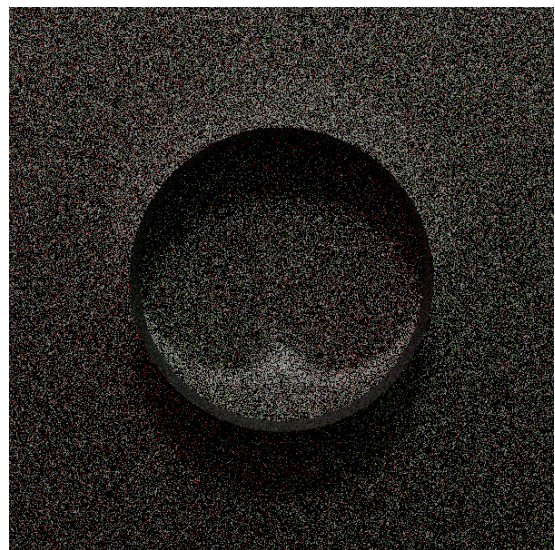
PT: 4 spp



CLT: 4 spp

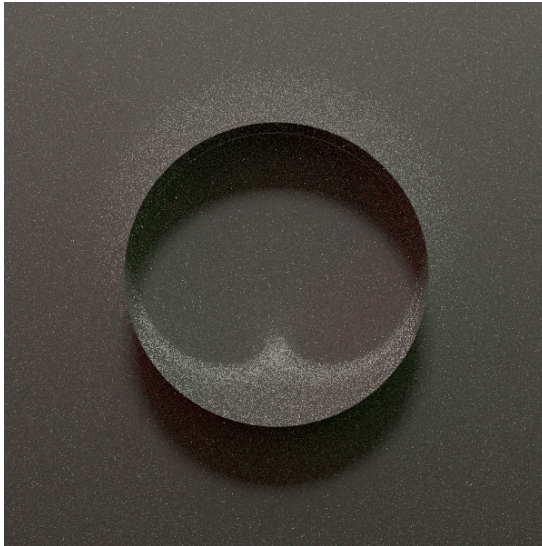


PT: 16 spp

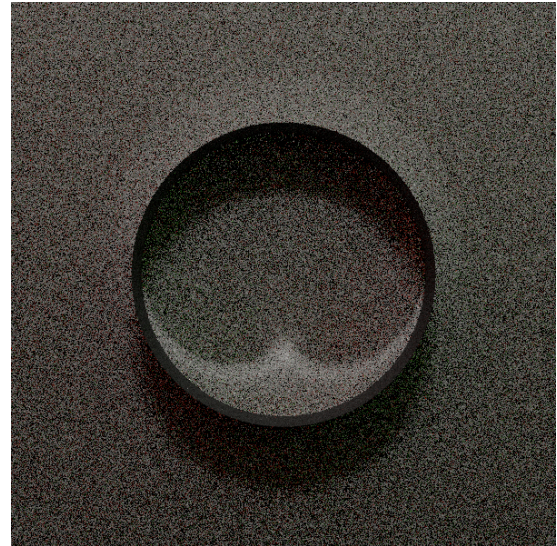


CLT: 16 spp

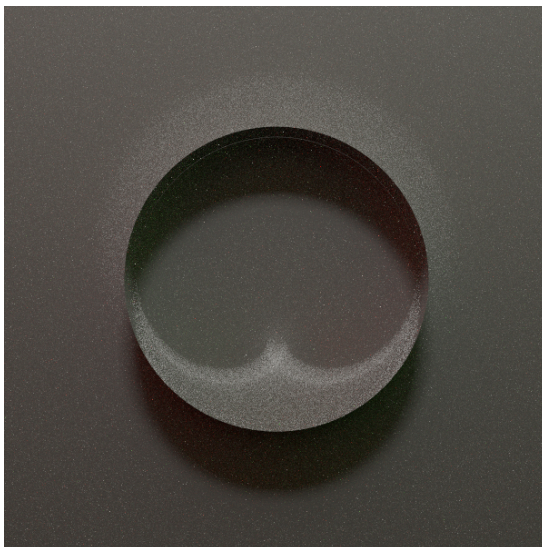
Figure 5.11 The ring rendered with PT and CLT at 1, 4, and 16 spp



PT: 64 spp



CLT: 64 spp



PT: 256 spp



CLT: 256 spp

Figure 5.12 The ring rendered with PT and CLT at 64 and 256 spp

fractive caustics in the glass sphere test case, though its performance for reflective caustics proved comparable to traditional path tracing. This difference stems from CLT’s current implementation using cosine-distributed importance sampling, following Lambert’s law for light emission. While this efficiently captures the energy distribution of area lights, it remains suboptimal for generating focused caustic patterns.

To better target caustic formation, the sampling strategy could be modified to explicitly prioritise specular surfaces - an approach analogous to next event estimation in path tracing, but operating in the opposite direction (light-to-camera rather than camera-to-light). Such an adaptation would more effectively exploit light paths contributing to bright caustic features, dramatically reducing the required sample counts for equal quality. Preliminary analysis suggests this optimisation could potentially reduce computation time by an order of magnitude for caustic-dominated scenes, bringing high-quality caustic rendering comfortably within real-time performance budgets on modern GPU hardware.

This strategic modification preserves light tracing’s inherent advantages - its unbiased nature and avoidance of complex data structures - while addressing its current limitations for certain caustic types. The resulting hybrid approach combines the conceptual simplicity of light tracing with sampling optimisations specifically designed for specular light transport phenomena.

5.3 CandelaDXR

Building upon the findings from the initial study, which identified light tracing as a viable technique for real-time caustic rendering due to its computationally efficient architecture and inherent suitability for GPU acceleration, this section presents an extended implementation within the CandelaDXR framework. Our approach incorporates a specialised light tracing shader developed using DXR, designed to achieve high-performance caustic rendering while maintaining physically accurate results.

The implementation process revealed several technical nuances and challenges, each systematically addressed through our development pipeline. The Anvil debugging framework proved instrumental in this regard, enabling early identification and resolution of tracing artefacts and sampling inconsistencies. To ensure rigorous assessment, we employ a comparative methodology, benchmarking our GPU implementation against reference renders generated by the verified CPU-based light tracer. This evaluation encompasses both qualitative assessment of visual fidelity and quantitative analysis of performance metrics, including computation times across various test scenes.

5.3.1 Design

Materials

Material sampling follows a hierarchical decision process based on the CandelaDXR material model, which extends the original Candela framework by incorporating Fresnel effects for physically accurate throughput and diffusion after transmission to model translucency. The throughput components are defined as $T_s = F_r(\text{dot}(x_0, n), n_1, n_2)$ for specular reflection, $T_d = (1 - F_r)d$ for diffuse reflection, and $T_t = (1 - F_r)(1 - d)$ for transmission, where d is the dissolve factor controlling material opacity. The sampling process begins by generating a random variable u_0 . This value is compared against T_s for specular reflection. If $u_0 \leq T_s$, the path is reflected specularly. If not, a second random variable v_0 is drawn. This value is then compared against the dissolve factor d to determine the secondary scattering event: if $v_0 \leq d$, the path undergoes diffuse scattering, further sampled from a cosine-weighted hemisphere; otherwise, it refracts according to Snell's law.

Figure 5.13 illustrates how an incoming ray (depicted in red) is split according to the CandelaDXR material model. The red ray intersects the material, and the Fresnel equations are used to determine the proportion allocated to the reflected ray (top blue arrow), with throughput F_r . The proportion of light that is transmitted (top orange arrow) and diffusely reflected (top green arrows) corresponds to $1 - F_r$. The dissolve value d determines how this throughput is distributed between diffuse reflection and transmission: a value of 1 allocates all throughput to diffuse reflection, while a value of 0 assigns it entirely to transmission. At the next intersection, which occurs within the material (after accounting for absorption via Beer's law), Fresnel is again applied to determine the allocation of throughput between total internal reflection (bottom blue arrow) and transmission. A transmission event is further subdivided into a specular component (bottom orange arrow) and a diffuse component (bottom green arrows), contributing to the modelling of translucency.

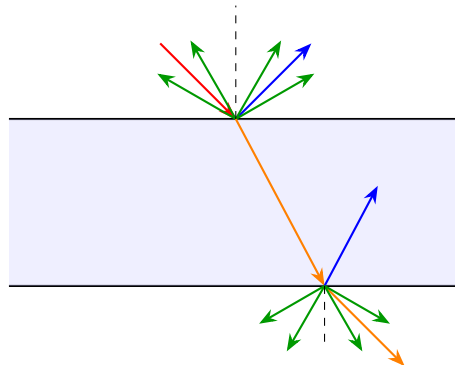


Figure 5.13 CandelaDXR Material. Rays: red is incoming, green is diffuse, blue is reflection and orange is refraction.

Pseudo-Random Number Generator

High-quality random number generation is critically important in light tracing implementations, as shaders relying on stochastic sampling processes are particularly vulnerable to artefacts when random number quality is suboptimal. Light tracers exhibit greater sensitivity to these statistical imperfections, given their dependence on direct sampling of light sources. When the underlying distribution lacks proper statistical properties, visible artefacts emerge, most notably on diffuse surfaces where sampling irregularities manifest as conspicuous patterns. As demonstrated in Figure 5.14, the use of an inadequate random number sampler produces wave-like artefacts along the walls of the Cornell Box scene, underscoring the necessity of robust random number generation in light transport simulation.

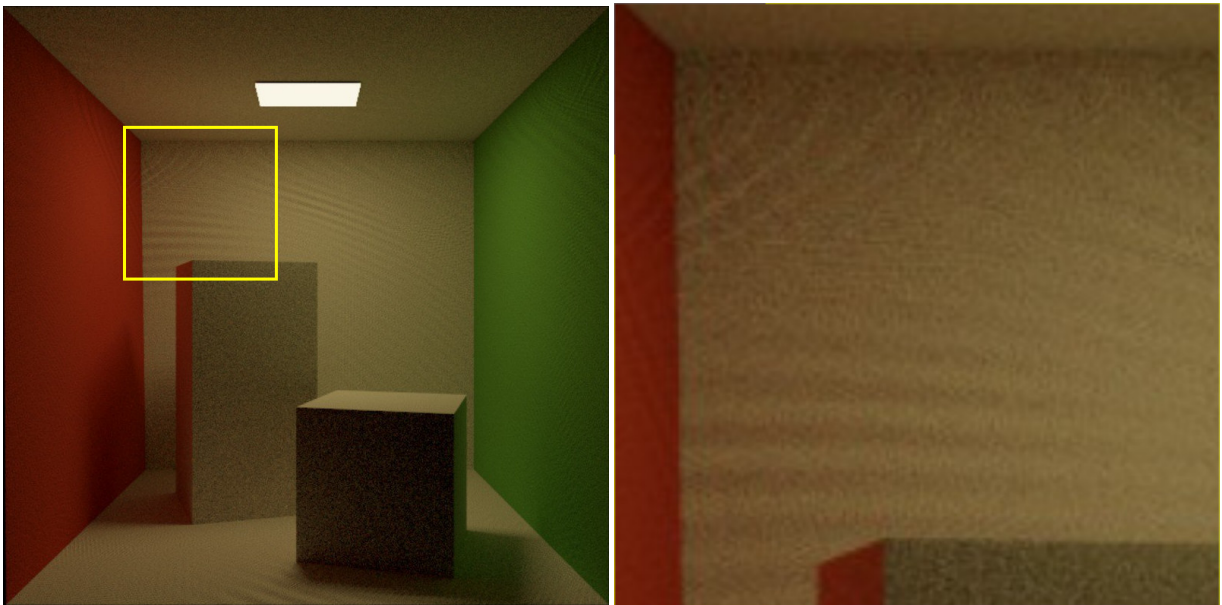


Figure 5.14 Poor sampler usage in Light Tracer. LCG (1664525, 1013904223). Artefacts can be observed as wave patterns

Our light tracing implementation utilises a permuted congruential generator (PCG) [99] as its foundational random number generation engine. The selection of PCG was motivated by its dual advantages of computational efficiency (requiring only minimal state storage) and native support for multiple independent streams, making it particularly well-suited to GPU-accelerated rendering. We employ the *pcg32i-random_t* variant, which provides strong statistical properties including a substantial 2^{32} period length and access to 2^{31} statistically distinct streams. Each stream guarantees both reproducibility (generating identical sequences given the same seed) and uniqueness relative to other streams.

Implementation-wise, we initialise all parallel threads with a common seed while ensuring stream uniqueness through thread-specific identifiers. The complete state of

each PRNG instance is persistently stored within a dedicated texture buffer, maintaining pixel-isolated computational states across renders. This architecture delivers three key benefits simultaneously: elimination of thread synchronisation overhead through completely independent execution paths, guaranteed render reproducibility via deterministic sequence generation, and statistically robust sampling outcomes through guaranteed stream independence. The texture-based state preservation further optimises memory access patterns for GPU architectures, ensuring efficient utilisation of parallel processing resources during light transport simulation.

Light Sampling

In light tracing, where rays are traced from the light sources, a sampling strategy must be established for each light source. Point lights are straightforward to sample, as points can be uniformly selected across the unit sphere. The probability of choosing any given point is $1/4\pi$. For spotlights, a similar approach could be employed by using rejection sampling to only accept points within the specified solid angle (lobe). However, this method would be inefficient, particularly when the lobe is narrow. Instead, direct sampling of the spherical cap is preferable. The probability density in this case is $1/(2\pi(1 - \cos(\phi)))$, where ϕ denotes the angular spread of the cone. This expression corresponds to the inverse of the spherical cap's surface area. In our implementation, the spotlight is modelled with a single lobe, omitting the distinction between an inner and outer cone.

Sampling directional lights presents a challenge, as they are typically modelled such that any unoccluded path from the shading point to the sky results in illumination. However, from the perspective of direct light sampling, this would entail an infinite area, making uniform sampling impractical. To address this, we first constrain the directional light to a finite region. This is achieved by constructing a bounded area above the scene, derived from the scene's axis-aligned bounding box (AABB) and the directional light's angle. We then sample this area uniformly with a probability density of $1/A$, where A denotes the area, while ensuring that emitted light follows a single fixed direction.

For area lights, the sampling process involves first selecting a random point on the light's surface, followed by generating a direction over the hemisphere. Given that area lights possess a defined surface element, we adhere to Lambert's cosine law, sampling the hemisphere using a cosine-weighted distribution rather than a uniform one to better match the energy distribution. The resulting probability density function is given by $A \cos(\phi)/\pi$, where A represents the area of the light and ϕ denotes the angle between the surface normal and the sampled direction.

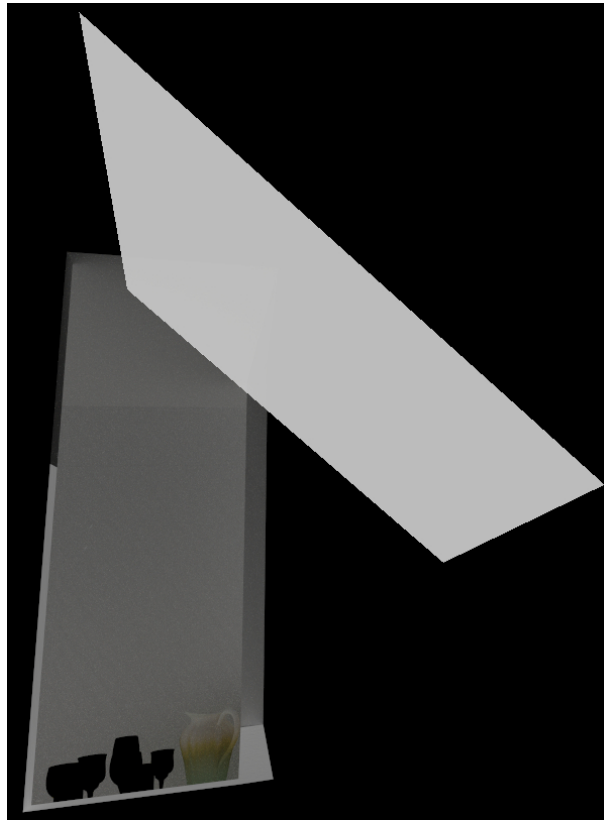


Figure 5.15 Visualisation of constrained directional light above scene

Data Structures

The light tracing implementation employs several specialised data structures distinct from those used in conventional shaders. Central to the algorithm is an irradiance accumulation texture (32 bytes per pixel), which stores two distinct components: diffuse illumination and caustic contributions, each represented as RGB values with an unused alpha channel for memory alignment. This separation facilitates potential post-processing operations in future extensions. The texture utilises unsigned integer storage to circumvent racing conditions inherent in multi-threaded floating-point accumulation. As shown in Listing 6, the underlying structure comprises two uint4 components (diffuse and caustic), with the accumulation buffer being cleared between frames to ensure proper temporal isolation.

Listing 6 *IrradianceItem* struct in C

```
struct IrradianceItem
{
    uint4 diff;
    uint4 caust;
};
```

Complementing this, a read-only conversion texture (4 bytes per pixel) containing a single floating-point value enables efficient irradiance-to-radiance transformation during the secondary rendering pass. This static texture requires regeneration only when the output resolution changes, maintaining computational efficiency.

The final output is maintained across frames through two persistent radiance accumulation textures (RGBA float format) - one dedicated to diffuse components and another for caustics. These buffers preserve their state throughout rendering sequences, only being invalidated during scene modifications such as animation updates.

Finally, to maintain the random number generation state across rendering operations, we employ a dedicated texture buffer that preserves the PRNG state for each pixel. This state texture utilises a compact representation, storing each pixel's current PRNG state as a single 32-bit unsigned integer (4 bytes per pixel). The state texture is persistently maintained between frames.

5.3.2 Concurrency Considerations

Several challenges were encountered during the implementation of the GPU light tracer. One such challenge pertains to the accumulation of the irradiance buffer. GPU threads operate most effectively when each thread writes to its designated memory. For instance, in path tracing, radiance is accumulated for each pixel independently, thus associating each thread with a specific pixel. Conversely, a light tracer begins by tracing light from a light source. A pixel receives irradiance based on the path taken by the light ray, a process characterised by randomness since light tracing utilises Monte Carlo integration. Therefore, when the process of light tracing is spawned, each thread is associated with a random light source and a random light path.

Accumulation in the irradiance buffer is not possible without synchronisation or atomic primitives since many threads may write to the same memory location (pixel). DirectX provides atomic primitives that allow for atomic operations on shared memory. Our implementation makes use of *InterlockedAdd*, a function that adds a reference to the destination and accumulates it with a given value. The main issue is that *InterlockedAdd* works only on integer types and as far as we know, there is no alternative that works with floating point types.

A workaround has been implemented but with some limitations. The floating point value is converted to an unsigned integer using $\min(\text{round}(\text{value} * \text{pow}(2, \text{rangeBits})), 2^{32})$. *rangeBits* is configurable and is set to 24 by default. This effectively encodes the floating point number to fixed point, allocating 8 bits to the integer part and 24 bits to the fractional part. This fixed point value can be accumulated normally using *InterlockedAdd* on the integer resource. This workaround is not perfect but it allows for accumulation of irradiance values. The reason is that small values could

underflow and very large values could potentially overflow. The effective range is from 2^{-24} to 2^8 . While this range is usable, there may be problems for caustics. Since caustics often concentrate light, there may be a large number of threads accumulating irradiance on the same pixel. Scenes having very bright light sources need more bits in the integral part, otherwise overflow occurs. Figure 5.16 shows the caustics produced by a scene (veach-bidir) where very bright caustics are produced. Distortion in the colour space occurs due to overflow because the integral part is only 8 bits. Shifting the balance to 16-bit for both integral and fractional parts solves the issue, this result is shown in Figure 5.17. In general, however, using a range of 8 bits works because most scenes do not define very bright light sources.

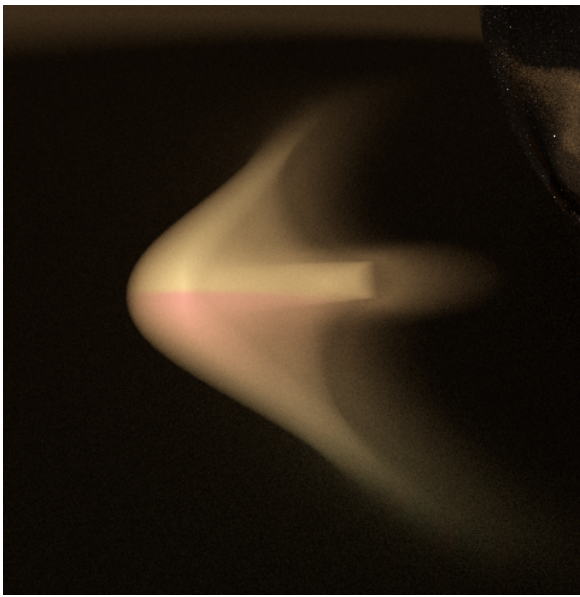


Figure 5.16 24-bit Range

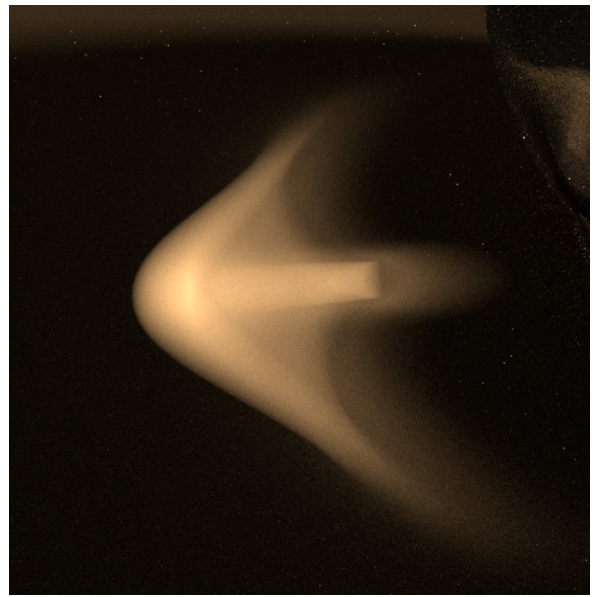


Figure 5.17 16-bit Range

5.3.3 Naive Light Tracer Pipeline

The light tracer is implemented via a ray-tracing-based pipeline, initiated by a specialised shader (depicted in Algorithm 3, with variable definitions in Tables 5.1, 5.2 and 5.3) that begins by selecting a random light source L from the set $\mathcal{L}$ and sampling a random emission direction, yielding an origin $\mathbf{x}_0$ and direction ω_0 . The initial contribution $\mathbf{C}$ is weighted by the light's flux Φ_L and the reciprocal of the sampling probability $p(\omega_0)$, propagating light throughout the scene iteratively. As the ray traverses the scene, at each intersection point $\mathbf{x}_i$, the material properties $\mathcal{M}$, including refractive index $\eta_{\mathcal{M}}$, diffuse reflectance ρ_d , dissolve factor d , and absorption coefficient σ_a are queried. The Fresnel function $\mathcal{F}$ computes the specular reflection proportion F based on the incident angle and the indices of refraction (η_1 and η_2 , assigned based on whether the hit is internal or external). For internal hits, Beer's law attenuates the contribution by $e^{-\sigma_a t_i}$.

Upon encountering a diffuse surface, the path's contribution to the pinhole camera is evaluated by checking visibility along direction ω_e and, if unoccluded, the computed radiance I , which incorporates the diffuse BRDF $f_r = \frac{\rho_d}{\pi}(1 - F)d$, geometric factors, and inverse square distance, is atomically accumulated into the corresponding pixel in the output image using the previously discussed synchronisation primitive. The path then continues by sampling a new direction ω_i through specular reflection, cosine-weighted diffuse scattering, or refraction via Snell's law, with probabilities proportional to F , $d_M(1 - F)$, and $(1 - d_M)(1 - F)$, respectively, ensuring energy-preserving transport. To ensure fair comparison with the path tracer, the total number of primary rays generated matches the product of the image's horizontal and vertical resolution dimensions.

To conclude the process, a final compute shader pass is executed to normalise the accumulated irradiance by dividing by the total number of samples. This averaging operation yields the estimated radiance per pixel, computed using Equation 5.14 as the fundamental relationship between accumulated measurements and final radiance values.

Variable	Definition
E	Irradiance contribution to be added to buffer
F	Fresnel reflectance coefficient
$\mathcal{F}$	Fresnel function
c	Sign factor for normal orientation (1 for external, -1 for internal)
d_m	Dissolve value. 0 for pure transmission, 1 for pure diffuse reflection
n_g	Geometric/flat normal at intersection
n_i	Interpolated/shading normal at intersection
n_{cam}	Camera normal/direction
p_{emit}	Full PDF of light emission sampling
u	Random variate for Russian roulette/material selection
ξ	Vector of random variates for sampling
η_1, η_2	Indices of refraction for current and adjacent media
η_m	Material index of refraction
ω_{cam}	Direction from intersection point to camera
ω_0	Initial emission direction from light source
ω_i	Next ray direction after scattering
ρ_d	Diffuse reflectance coefficient

Table 5.3 Variable Definitions Specific to Algorithm 3

5.3.4 Methodology

The light tracer is executed on a Windows 11 system equipped with an Intel i7-8700K CPU and an NVIDIA GeForce RTX 2080 Ti GPU. A varied selection of scenes is employed, incorporating diverse lighting conditions and materials to facilitate the generation of distinct caustic patterns, including both low- and high-frequency caustics. Per-

Algorithm 3 CandelaDXR Light Tracer

```

1: procedure LIGHTTRACEDXR
2:    $\xi \sim U(0, 1)^*$  ▷ Random variates
3:    $P \sim \text{ChooseLight}(\xi)$  ▷ Select light source from  $\mathcal{L}$ 
4:    $T \leftarrow |\mathcal{L}|$  ▷ Initial throughput
5:    $(x_0, \omega_0, p_{\text{emit}}) \leftarrow \text{SampleLightEmission}(P, \xi)$  ▷ Returns position, direction, and full PDF
6:    $T \leftarrow T \cdot \frac{L_e(x_0, \omega_0)}{p_{\text{emit}}}$  ▷ General emission sampling
7:    $i \leftarrow 1$ 
8:   while TraceRay( $x_{i-1}, \omega_{i-1}$ )  $\rightarrow (x_i, t_i, n_i, n_g, m)$  do
9:      $T \leftarrow T \cdot \left| \frac{\omega_{i-1} \cdot n_i}{\omega_{i-1} \cdot n_g} \right|$  ▷ Handle normals.  $n_i$  interpolated,  $n_g$  flat/geometric
10:     $\eta_1, \eta_2, c \leftarrow \begin{cases} (\eta_m, 1, -1) & \text{if internal} \\ (1, \eta_m, 1) & \text{otherwise} \end{cases}$ 
11:     $F \leftarrow \mathcal{F}(-c(\omega_{i-1} \cdot n_i), \eta_1, \eta_2)$ 
12:    if internal then  $T \leftarrow T \cdot e^{-\sigma_a t_i}$ 
13:    end if
14:     $\omega_{\text{cam}} \leftarrow \frac{x_{\text{cam}} - x_i}{\|x_{\text{cam}} - x_i\|}$ 
15:    if  $\omega_{\text{cam}} \cdot n_i > 0 \wedge \omega_{\text{cam}} \cdot n_{\text{cam}} > 0$  then
16:       $p \leftarrow \text{ProjectToPixel}(x_i)$ 
17:      if  $\neg \text{Occluded}(x_i, \omega_{\text{cam}})$  then
18:         $f_r \leftarrow \frac{\rho_d}{\pi} (1 - F) d_m$ 
19:         $E \leftarrow T \cdot f_r \cdot \frac{(\omega_{\text{cam}} \cdot n_g)(\omega_{\text{cam}} \cdot n_{\text{cam}})}{\|x_{\text{cam}} - x_i\|^2}$ 
20:        AddIrradianceAtomic( $p, E$ )
21:      end if
22:    end if
23:     $u \sim U(0, 1)$ 
24:    if  $u \leq F$  then ▷ Specular reflection
25:       $\omega_i \leftarrow \text{reflect}(\omega_{i-1}, cn_i)$ 
26:    else if  $u \leq F + d_m(1 - F)$  then ▷ Diffuse reflection
27:       $\omega_i \leftarrow \text{SampleCosineHemisphere}(n_i)$ 
28:       $T \leftarrow T \cdot \frac{\rho_d(\omega_i \cdot n_i)}{\pi p(\omega_i)}$ 
29:    else ▷ Transmission
30:       $\omega_i \leftarrow \text{refract}(\omega_{i-1}, cn_i, \eta_1/\eta_2)$ 
31:    end if
32:     $i \leftarrow i + 1$ 
33:  end while
34: end procedure

```

formance evaluation is conducted by measuring the time required to render a single frame, expressed in milliseconds per scene, thereby establishing the permissible computational budget. The timing measurement is restricted to the light tracing phase using DirectX Timestamp Queries (*D3D12_QUERY_TYPE_TIMESTAMP*), encompassing the clearing of render target views for diffuse and caustic buffers, the light tracing pass, the irradiance-to-radiance conversion via compute shader, and the compute shader responsible for transferring radiance output from each buffer (diffuse and caustics) to subsequent pipeline stages. To enhance statistical reliability, the recorded timings are averaged across multiple frames.

Quantitative image quality assessment is performed using HDRVDP3, comparing the noisy output of each scene against a reference image generated by the same light tracer with substantially higher per-pixel sample counts, focusing exclusively on the caustics buffer. Furthermore, rendered scenes are exhibited at low sampling rates to visually illustrate noise characteristics, facilitating qualitative analysis to support subsequent discussion.

The results obtained from these experiments inform the subsequent methodological refinements. The key considerations centre on whether the current rendering performance and noise levels meet the required thresholds, or if further optimisation is necessary. Given that denoising introduces inherent bias, potentially compromising physical accuracy, strategies to mitigate this trade-off must be evaluated.

5.3.5 Evaluation

The reference images for scenes used in this evaluation are presented in Figure 8.5. These images were rendered using hybrid rendering, which combines path tracing and light tracing, at approximately 4 million spp. Light tracing techniques with importance sampling, as discussed in Chapter 8 and detailed in Table 8.2, were also employed. This configuration applies only to the reference images; the noisy images analysed subsequently use pure light tracing without these importance sampling techniques. Path tracing was used for paths that the light tracer cannot generate, specifically those connecting specular objects with the pinhole camera. It is important to note that caustic buffers, which are central to this study, depend entirely on light tracing, since no paths generated by path tracing contribute to them.

Figure 5.18 presents the caustics buffers for all scenes rendered at 16 spp, demonstrating significant noise across all cases. While discernible caustic patterns emerge in CUP, DICE, OUTER, TORUS, EGG, and WATER, the CUPS and REFLECT scenes exhibit excessive noise that obscures their caustic structures.

Performance metrics in Table 5.4 reveal that six scenes achieve efficient rendering times, averaging approximately 2.1 ms per frame (1 spp). However, EGG (8.38 ms) and

`WATER` (5.02 ms) demonstrate notably slower performance. The global average of 3.27 ms remains well within the computational budget for real-time 60Hz rendering.

Quality assessment via HDRVDP3 in Figure 5.19 compares 16 spp renders against high-reference images (at least 4 million spp). While `CUP`, `TORUS`, and `WATER` display the expected monotonic quality improvement, other scenes exhibit an initial decrease followed by increasing scores in early sampling stages, suggesting non-linear convergence characteristics.

scene	lt (ms)	scene	lt (ms)
CUP	2.11	OUTER	2.65
CUPS	2.06	TORUS	1.80
REFLECT	2.06	EGG	8.38
DICE	2.10	WATER	5.02

Table 5.4 LT Shader performance per frame (approximate in ms) per scene.
Hardware: Intel i7-8700K, NVIDIA GeForce RTX 2080 Ti

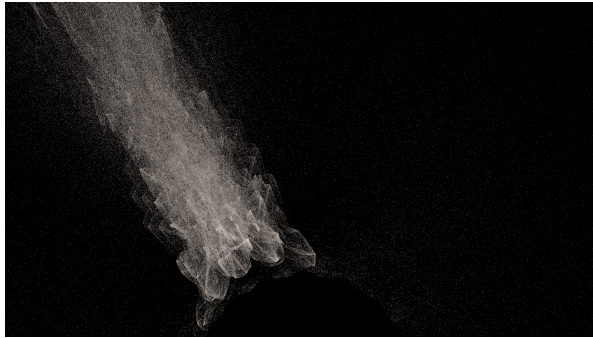
5.3.6 Discussion

The evaluation results demonstrate that the light tracing shader achieves an average execution time of 3.27 ms per frame, theoretically enabling 5 spp when accounting for parallel execution with other shaders utilising different GPU resources. This sample count could potentially be increased by optimising the shader to process multiple samples per invocation, thereby eliminating redundant buffer copies and irradiance-to-radiance conversions for each sample.

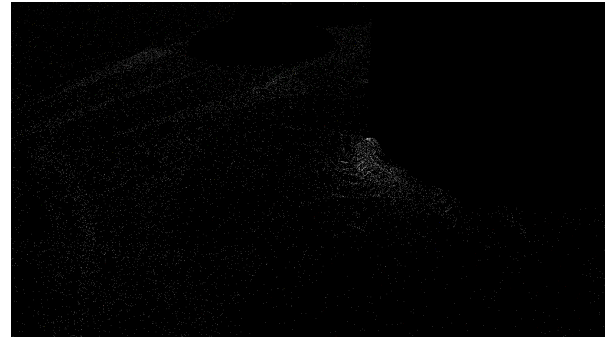
Hardware limitations also play a significant role in performance. The current implementation runs on an NVIDIA RTX 2080 Ti (2018), which features 68 ray tracing cores. In comparison, modern GPUs such as the RTX 5090 offer 170 ray tracing cores, representing a theoretical 2.5-fold performance increase, excluding architectural improvements per core. This suggests that 16 spp could be feasible on newer hardware.

However, at 16 spp, the HDRVDP3 scores reveal an unexpected decline in several scenes, likely due to persistent noise overwhelming the signal despite increasing sample counts. This phenomenon is examined in greater detail in Chapter 8.

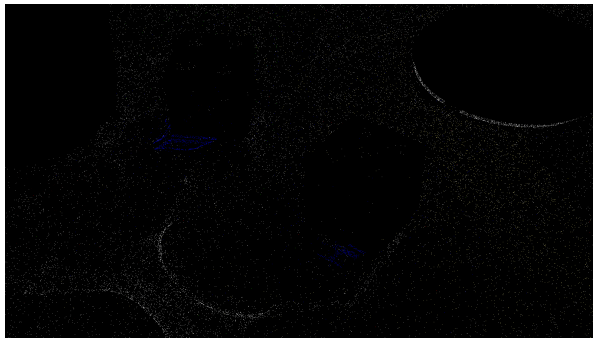
While light tracing successfully generates caustics at low sample counts, the raw output remains unusable without post-processing. For simpler cases like `CUP`, where high-frequency caustics dominate, a basic high-pass filter may suffice. However, scenes such as `CUPS` and `REFLECT` exhibit noise intricately coupled with the caustic signal, demanding more sophisticated denoising approaches. Further research is needed to develop real-time caustics processing methods that balance noise reduction with signal preservation.



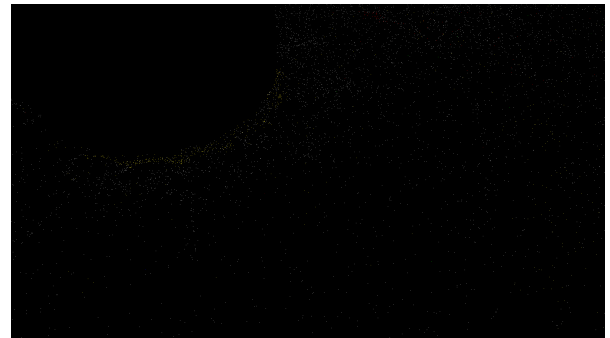
(a) CUP



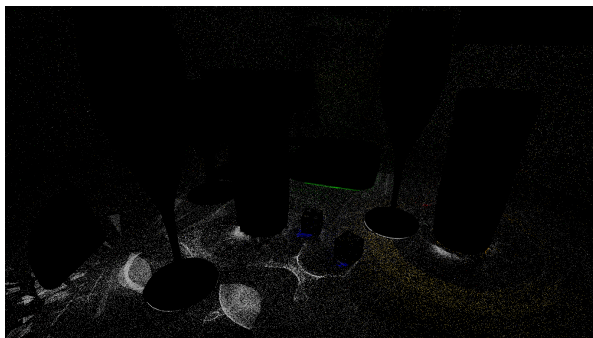
(b) CUPS



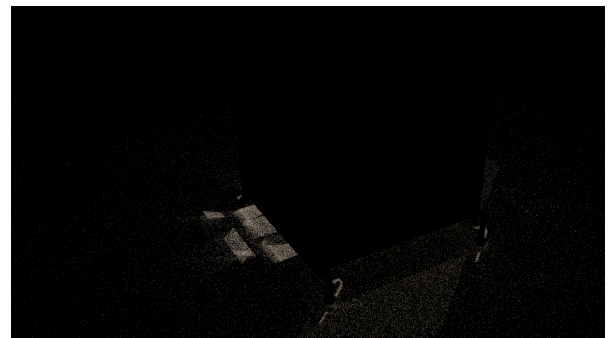
(c) DICE



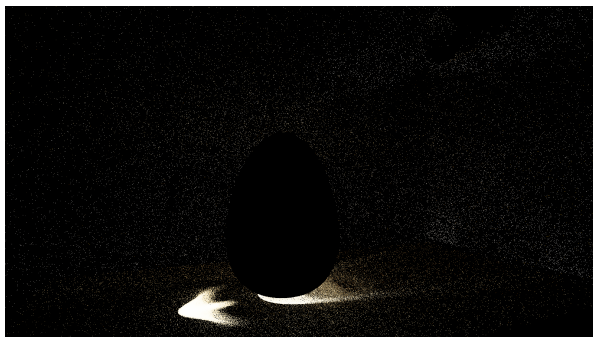
(d) REFLECT



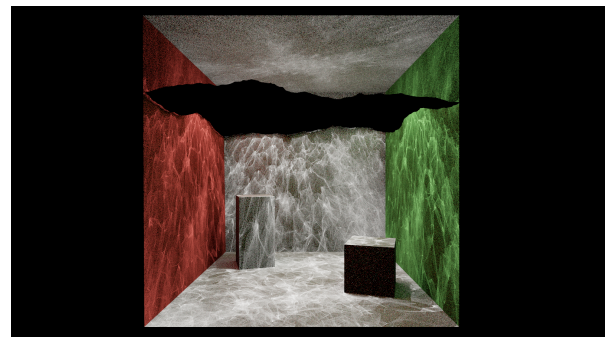
(e) OUTER



(f) TORUS

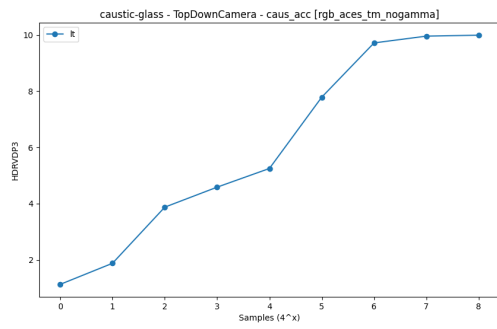


(g) EGG

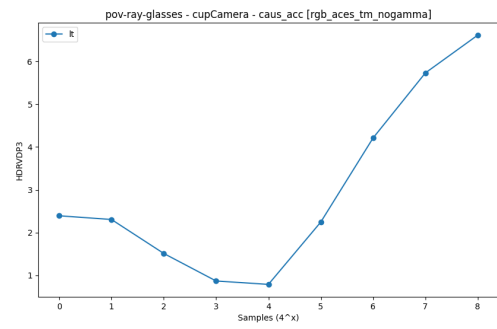


(h) WATER

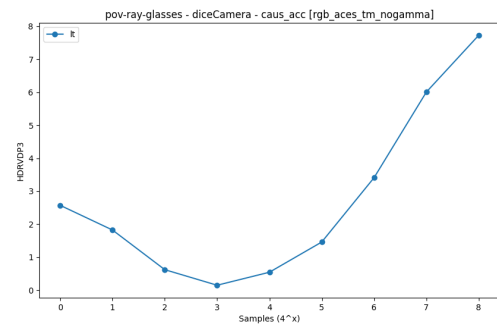
Figure 5.18 Caustics Buffers at 16 spp



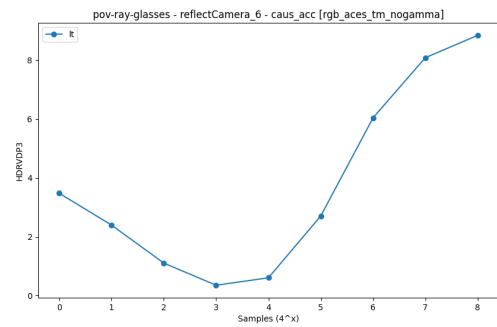
(a) CUP



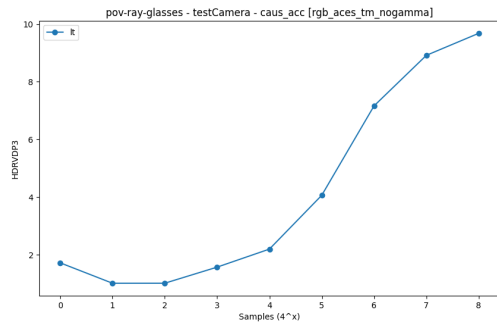
(b) CUPS



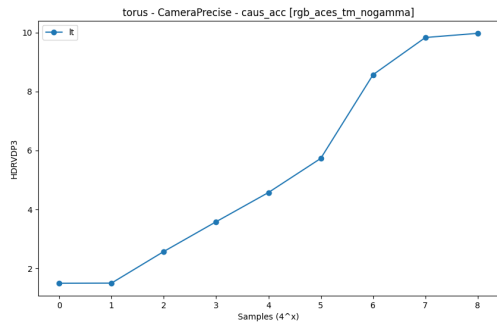
(c) DICE



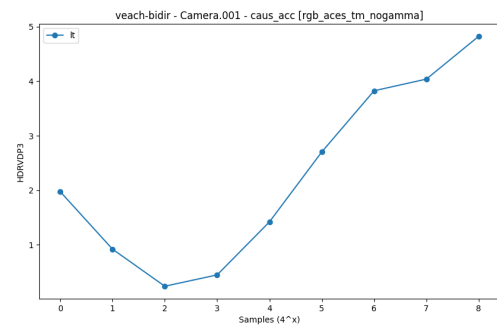
(d) REFLECT



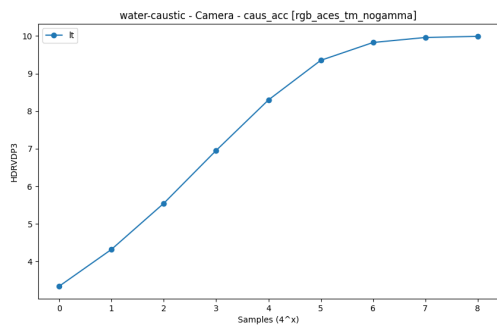
(e) OUTER



(f) TORUS



(g) EGG



(h) WATER

Figure 5.19 HDRVDP3

Alternative approaches for improving signal quality at the sampling stage prior to post-processing should be investigated. However, such improvements typically entail increased computational costs, necessitating a careful balance between convergence speed and perceptual quality. Future work should explore adaptive sampling strategies or hybrid rendering techniques that selectively enhance critical regions while maintaining real-time performance constraints. The trade-offs between computational overhead and visual fidelity must be systematically evaluated to determine optimal sampling configurations for practical applications.

5.4 Summary

This chapter presented a comprehensive study of light tracing (LT) for rendering caustics, demonstrating its effectiveness in handling complex light paths that are challenging for traditional path tracing. The implementation of a CPU-based light tracer (CLT) provided a reference for verifying the algorithm’s correctness, while the GPU adaptation leveraged the computational power of modern hardware to achieve real-time performance. Key findings include LT’s superior efficiency in rendering refractive caustics compared to path tracing, though its performance for reflective caustics was comparable. The evaluation highlighted LT’s strengths in handling high-frequency light transport, particularly in scenes dominated by caustics, while also identifying limitations in noise management and sampling efficiency. These insights set the stage for further optimisation, particularly in denoising and hybrid rendering techniques, which will be explored in subsequent chapters. The results underscore LT’s potential as a viable solution for real-time caustic rendering, especially in applications where accuracy and performance are critical. The development of an evaluation framework, such as the one used in this study, would significantly streamline future research by providing a standardised platform for testing and comparing rendering techniques.

6 Forge

In this chapter we introduce Forge — an extension of the Anvil system initially presented as a visual debugging tool for PBR [100]. Anvil provides higher-level abstractions for common visual debugging tasks, operating on structured, reusable data types such as *atoms* and *molecules*, representative of essential rendering primitives like rays and vectors. Building upon this foundation, we substantially expand Anvil’s capabilities in this work, transforming it into a generalised system suitable for systematic, reproducible evaluation of computer graphics pipelines.

6.1 Challenges in Light Tracing and Denoising

In Chapter 5, we assessed the real-time performance of LT for rendering caustics, considering both the quality of the resulting caustics and its computational speed. Although promising results were obtained, the renders produced remained excessively noisy, limiting their practical application. Consequently, two main strategies emerged for enhancing the quality of renders produced by LT: optimising convergence or employing appropriate denoising strategies.

Improving convergence often involves integrating sophisticated algorithms from offline rendering, which invariably entails a trade-off between computational overhead and the required image quality. While increasing computational costs can partly alleviate noise issues, residual noise typically persists, suggesting that supplementary solutions are essential. Thus, specialised denoising approaches targeting caustics are particularly appealing.

General-purpose denoisers such as ReBLUR and ReLaX, available for real-time rendering applications, were integrated into our CandelaDXR pipeline. Although these methods effectively reduce noise, we observed a significant degradation of visual fidelity, particularly an unwanted loss of sharpness and clarity of the caustics themselves. This excessive blurring of caustics indicated that general denoising solutions were insufficiently tailored for rendering tasks concerning highly detailed caustics patterns.

A promising alternative is spectral denoising, a method involving thresholding coefficients within the spectral domain to suppress noise more selectively. We hypothesised that spectral-based denoising could more effectively isolate noise from signal, thereby achieving denoising without compromising the sharp definition that characterises caustics. Nevertheless, to leverage spectral denoising, one must determine the optimal combination of parameters (such as spectral algorithms, threshold values, tone mappers, and evaluation metrics) within a vast, combinatorial parameter space. Manually exploring and evaluating this parameter space in a systematic and reproducible

manner rapidly becomes cumbersome, tedious, and prone to human error.

This challenge motivated the need to systematically streamline the evaluation process. During our experimentation, several repetitive and commonly shared tasks emerged, such as comparing rendered images using multiple similarity metrics and applying standardised denoising algorithms to image buffers. Initially, ad-hoc scripting was used to combine these operations; however, such approaches quickly proved insufficient due to complexity, lack of standardisation, and the high risk of configuration mistakes. Beyond denoising, broader challenges in graphics evaluation further complicate systematic analysis.

6.2 Complexity in Graphics Software Evaluation

Software evaluation, especially within computer graphics, plays a fundamental role in determining correctness (for instance, whether a specific model like the rendering equation is being solved accurately), performance (encompassing both computational speed and output visual quality), and overall graphical fidelity within rendering engines. Unlike standard software performance evaluations, which are often measurable through straightforward metrics, graphics applications introduce evaluation complexities involving inherently subjective criteria such as visual quality and rendering realism. These criteria are far more challenging to quantify. Common metrics like PSNR, SSIM, and advanced perceptual difference methods are typically employed in the context of Monte Carlo rendering to assess a renderer’s capacity to accurately simulate phenomena such as real-world lighting, shading, and textures.

An acute challenge arises from the necessity for rigorous consistency when configuring rendering conditions across different systems or implementations. Even small variations in scene setup, camera positioning, or illumination can compromise comparisons and lead to misleading findings. This is particularly critical when generating reference images used to benchmark different rendering approaches. Manual interventions for ensuring consistency introduce significant risk of human error, posing a serious obstacle both to the reliability and the reproducibility of research workflows. Compounding this difficulty is the tight coupling that often develops between evaluation logic and specific rendering applications, which reduces independence and reusability of evaluation components.

6.3 Introducing Forge

Forge inherits Anvil’s modular and extensible architecture, effectively decoupling evaluation tasks from specific rendering engines and providing reusable, standardised work-

flows. With Forge, researchers not only can perform detailed, consistent, and reproducible evaluations, but also have the flexibility to compare rendering approaches readily across distinct systems without being burdened by manual configurations and tool-specific constraints. Moreover, Forge introduces a robust WebSocket-based evaluation framework that supports decentralised, distributed execution, even across multiple platforms and programming languages. This design considerably enhances scalability, adaptability, and usability, making Forge suitable for diverse scenarios ranging from quick, iterated experiments to complex distributed render farms.

The case studies presented in this chapter demonstrate Forge’s capabilities in verifying rendering techniques, comparing their performance, and enabling the development of new algorithms. By integrating evaluation workflows, we contribute a powerful tool for both research and development in the field of computer graphics, streamlining the process of testing and validating rendering software across a variety of applications.

6.4 Forge - An Anvil Evaluation Framework

Forge is a superset of Anvil, extending its functionality while preserving its core design principles. It is intended to be simple, granting access to powerful analytical tools with minimal setup and disruption. These analytical tools are implemented as systems within the ECS design pattern as Anvil supports extensibility through the addition of further systems, which can be registered via the plugin system.

Anvil operates as a C++ library; applications requiring debugging would link to it either statically or dynamically and submit a stream of entities with a list of relevant components. These components are well-defined and documented. For instance, to debug a path, an entity with a *path* component is needed. In Anvil’s terminology, this component is a *molecule* consisting of relevant *atoms*, such as position, direction, etc. Once all the necessary components for a system are present, the system in question becomes active for evaluation.

Whilst employing Anvil as a library proves effective for debugging purposes, it is considered less straightforward for the evaluation of rendering pipelines. To facilitate a more natural evaluation methodology, Forge executes as a standalone application. This design modification enables centralised control and permits the implementation of standardised evaluation workflows. In this operational mode, renderers must necessarily expose an interface compatible with paired Forge systems.

6.5 Design

Forge is an application framework that incorporates all of Anvil's functionality, except that it is responsible for executing the systems itself. Therefore, the component repository, system registry and entities are all stored and managed by Forge. The `tick` method is invoked to execute systems. `tick` iterates through all available and active systems sequentially in their order of registration, passes the entities as parameters and executes the systems. Throughout the chapter, `tick0` represents the first `tick` call, `tick1` represents the second `tick` call, etc. Forge requires startup configuration before evaluation starts. Figure 6.1 demonstrates a general configuration file that lists one evaluation, and for this evaluation, the Evaluation System is loaded. Optional configuration may be added for each individual evaluation.

```
{
  "evaluations": [
    {
      "name": "EvaluationSystem",
      "config": {
        "key": "value"
      }
    }
  ]
}
```

Figure 6.1 JSON configuration for a general evaluation system

Figure 6.2 illustrates the evaluation process which is typically divided into three phases. Following startup configuration, the *initialisation* phase (`tick0`) loads and initialises required systems. The evaluation system typically adds necessary entities and components during `tick0` or `tick1`. Between `tick1` and `tickn-1`, in the *evaluation* phase, systems perform their evaluation tasks where results are iteratively collected and processed - for instance, retrieving rendering images and then computing similarity scores. At `tickn`, in the *finalisation* phase, where the evaluation result is presented and cleanup occurs, the evaluation system removes all systems added during `tick0` and any entities with their components created during evaluation. If specified in the startup configuration, additional evaluations may follow. Evaluation systems are thus responsible for system configuration, bootstrapping, and data collection throughout this process.

6.5.1 Synchronisation

In Forge, systems are arranged in sequence, executing one after the other within one tick. However, systems that do not have data dependencies – that is, those that nei-

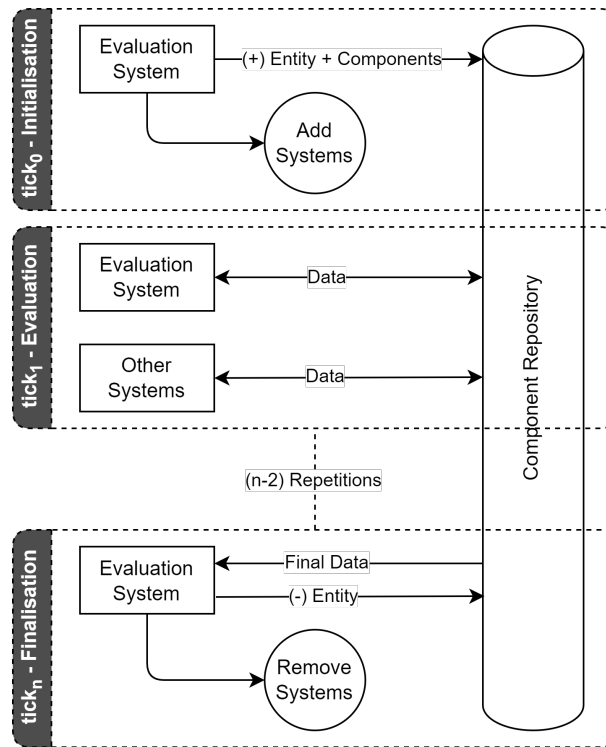


Figure 6.2 General process diagram for one evaluation. (+) is add, (-) is remove.

ther read from nor write to the same component types – can run in parallel to optimise throughput. Forge provides a synchronisation specification to ensure component consistency across multiple Forge instances. Any implementation of the specification (Synchronisation System) should be placed after systems for which entity synchronisation is required, acting as a fence that ensures all preceding dependent systems have completed their execution.

Figure 6.3 depicts Forge in a multi-process configuration. The Forge main instance is initialised with multiple systems: System set A (executes prior to the synchronisation block), the Synchronisation System, and System set B (a set of systems executing after the synchronisation block). System set C operates in a separate process, meaning memory is not shared. In this case, the synchronisation system has been improved to support decentralised systems, which may operate on the same or different machines. It ensures that System set A (local) are prepared before transmitting and receiving data from System set C on a separate Forge instance, which runs its own systems and modifies the entities as needed. System set B can proceed with their execution once synchronisation is complete.

6.5.2 WebSocket Synchronisation

The WebSocket System, designed to implement the synchronisation specification, has been created to facilitate integration with various systems, including those located re-

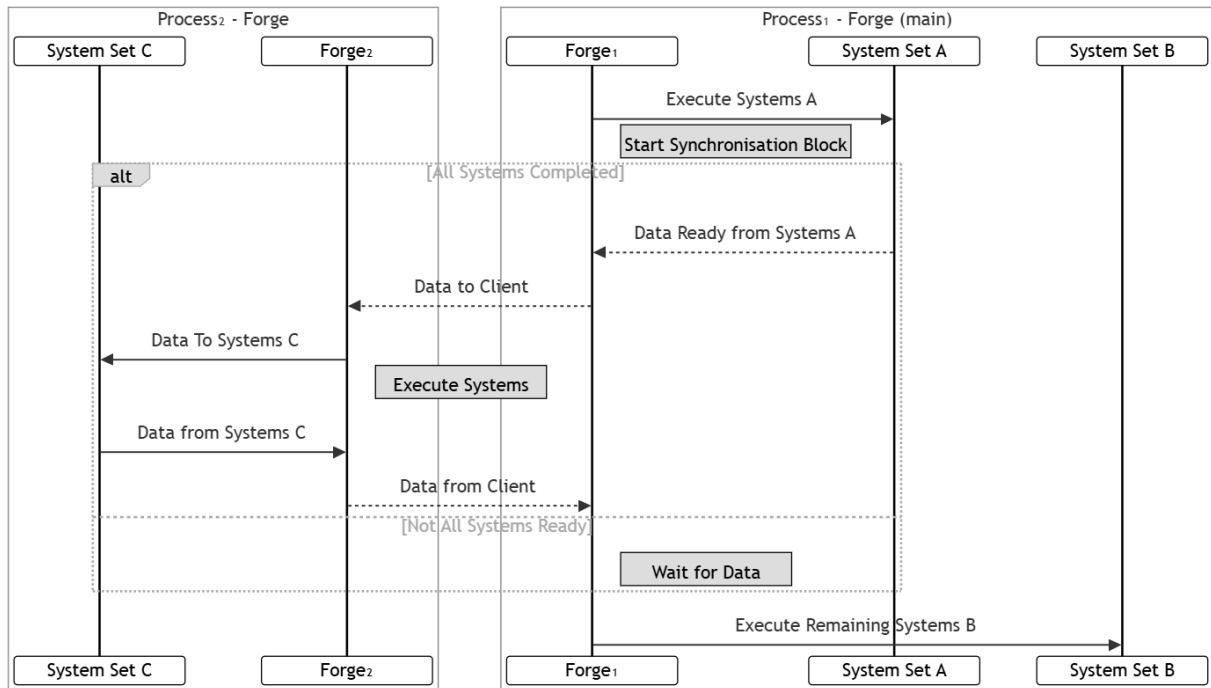


Figure 6.3 Multiple Forge instances communicating through the synchronisation specification. The synchronisation block (dashed rectangle) depicts the synchronisation process; the synchronisation systems are implicitly represented between System sets A and B, and encompassed in System set C.

motely. Forge can operate across multiple language runtimes (such as Python), leveraging libraries available within those environments. WebSockets were chosen for their broad compatibility with many programming languages and their ability to function in web browsers, allowing developers to build evaluation software with web technologies, speeding up the development of user interfaces.

The WebSocket System works in two modes: *server* and *client*. In *server* mode, it can be positioned between systems, capturing a snapshot of all entities at a given moment and broadcasting it to all WebSocket Systems operating in *client* mode. In *client* mode, the WebSocket System encapsulates other systems that require external data to function, with the encapsulated systems operating within a single, isolated context. This isolated context depends solely on the state of the entities maintained by the corresponding *server* instance.

Whenever the `tick` function is called, the *server* instance sends these entities to all connected *client* instances. Systems running on a *client* can modify these entities as needed. Once all systems on the *client* side have completed execution, the modified entities are sent back to the *server*, which updates the original entities, creating the effect that the changes occurred locally. However, entities that contain reflection components [100] are treated as read-only, and any attempt to modify them will be silently ignored.

6.6 Usage

Most of the implementation discussed below requires implementing a Forge system using its interface. The system interface requires two methods to be implemented; `execute()` and `required_components()`. When Forge calls `tick`, `execute()` is automatically invoked. The `required_components` method has been introduced by Forge and is optional. However, when implemented, it should return a list of components that the system requires. This functionality enables Forge to construct a dependency graph for the concurrent execution of systems.

Depending on the specific use case, a Proxy System may be required for any components or systems that are not inherently supported by Forge and are running in separate processes. This includes external tools, user-created library components, or other third-party systems that are distinct from those provided by Forge. For example, to integrate an external renderer with Forge for evaluation purposes, the following approach should be adopted:

1. **Proxy System Development:** An Anvil system should be developed to function as an intermediary between Forge and the renderer.
2. **Interface Requirement:** The renderer must expose an interface that the Proxy System can utilise for configuration purposes.
3. **Configuration Handling:** The Proxy System should be designed to accept standard renderer configuration molecules, which it will use to properly initialise the renderer.
4. **Result Collection:** The Proxy System should also collect the output from the renderer and organise it into molecules that Anvil can process.
5. **Direct Molecule Option:** For renderers that have already been integrated with Anvil for debugging purposes, there is an option to simplify the Proxy System. In these cases, the renderer can be modified to emit result molecules directly, in a format that is compatible with Forge.

When a required Evaluation System is not available, users must implement a custom one following the three stages described in Section 6.5: *initialisation*, *evaluation*, and *finalisation*. An Evaluation System operates as a state machine, with each stage triggered by `tick` events and synchronised across all systems.

This multi-stage approach is a consequence of the ECS pattern, ensuring that all systems perform the necessary work and remain synchronised at each stage of the evaluation.

Finally, once the implementation is complete, Forge instances are initialised using a configuration similar to that shown in Figure 6.1. In cases of multiprocess or distributed setups, each Forge instance must be loaded with its own JSON configuration file.

6.7 Case Studies

Forge was evaluated using two CG test cases: comparing outputs between two different rendering algorithms on the same scene, and assessing spectral denoising with pre-computed coefficients in animation. In addition to being typical tasks in CG evaluation workflows, such as rendering buffers and comparing raw versus denoised images, the use cases also highlight Forge’s reusability and synchronisation features.

6.7.1 Light and Path Tracing Verification

Verification is a critical aspect of computer graphics, particularly for ensuring the accuracy of rendering algorithms. When comparing different techniques, such as light tracing and path tracing, verification is essential to confirm the correctness of their implementations and the consistency of their outputs. It is crucial to verify that both results are either identical or, at the very least, closely aligned within acceptable error margins, which can be determined using image similarity metrics. This comparison serves several purposes: it confirms the theoretical equivalence of the algorithms, verifies the accuracy of their implementations, and helps identify any discrepancies that may arise due to factors such as numerical precision, sampling strategies, or other implementation-specific details.

In this experiment, we use our in-house renderer, CandelaDXR, which is operated via the command line by specifying a JSON configuration file that allows for the adjustment of various parameters. To integrate CandelaDXR with Forge, a corresponding Proxy System was developed. There are two main approaches to building this system. One option is to expose CandelaDXR as a shared library and develop the Proxy System in a language capable of interfacing with this library. However, modifying CandelaDXR’s code could introduce new bugs. The alternative is to write the Proxy System in any language, configuring and launching a new CandelaDXR process to extract the required results. Both methods depend on the renderer’s configurability being sufficiently expressive. In this case, spawning a new process for each approach is simpler and preferable, as computational performance is not a primary concern. We opted to implement this system in Python and it is referred to as the Candela System.

When an Entity contains a `RenderConfig` molecule, this System launches a CandelaDXR renderer process using a JSON configuration generated from the molecule,

Metric	Score	Score TM
FLIP	4.53e-02	8.12e-03
HDRVDP3	9.90	8.84
MSE	9.38e-05	1.77e-05
PSNR	69.94	47.53
SSIM	1.00	0.99

Table 6.1 Cornell Box Shiny comparing light and path traced scores. TM = ACES tone mapping. FLIP and MSE, lower is better.

and waits for the specified frame to be processed. This configuration dictates whether the light tracer or path tracer is used, along with the scene setup. Upon completion, CandelaDXR outputs the resulting buffers to the file system. The System then attaches the radiance buffer's contents as a `Buffer` molecule, which includes metadata detailing the buffer type (such as caustics or G-buffer), the pixel format, and whether the image is a reference or noisy.

The image comparison functionality is handled by a separate Python application (another Forge instance). This paired System captures noisy buffers and compares them either against a reference image or other noisy buffers. It retrieves entities containing the `Buffer` molecule, executes the image comparison, retrieves the results (which include five different metrics per comparison), and attaches these results as a new molecule, `Metrics`.

Lastly, the Renderer Evaluation system was developed. On startup, this system loads its configuration. For this experiment, both the path tracing and light tracing algorithms are evaluated using a modified Cornell Box scene. On `tick0`, this data is embedded in the `RenderConfig` molecule, and the WebSocket System is configured to

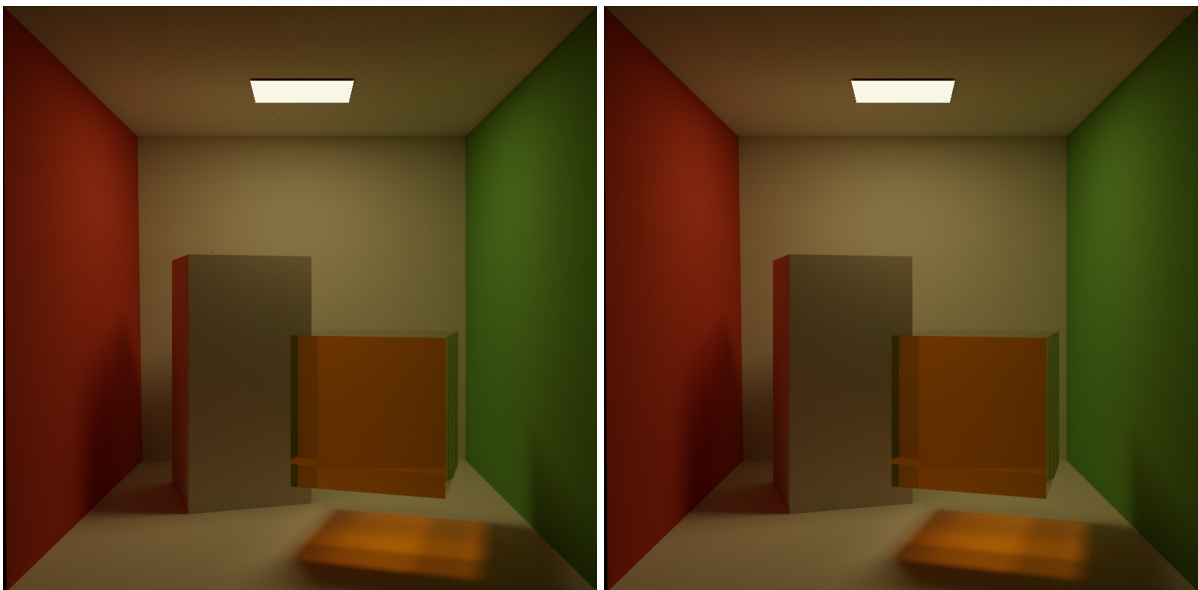


Figure 6.4 Left: Path traced, Right: Light traced

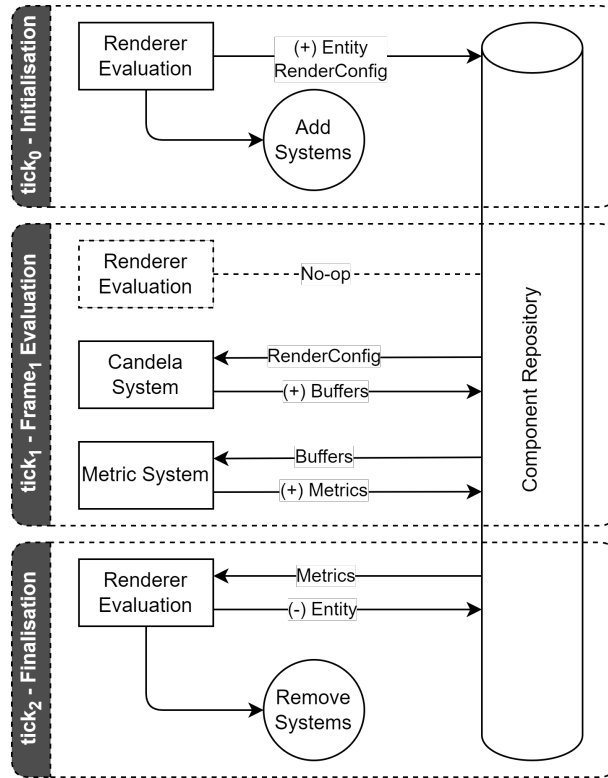


Figure 6.5 Light and path tracing flow diagram.

run in *server* mode, waiting for both the Candela and Metric systems to become operational. The Renderer Evaluation System then monitors for the appearance of the *Metrics* molecule on the initial Entity, signalling that the evaluation is ready. Once this occurs, the results are recorded (see Table 6.1), and the System performs clean-up.

In the other processes, the WebSocket System is set to *client* mode, wrapping the Candela and Metric systems. During tick_1 , the Candela System receives the *RenderConfig* molecule and generates the path and light tracing buffers, which are then attached to the same Entity and displayed in Figure 6.4. The Metric System processes these buffers and attaches the comparison results using a *Metrics* molecule on the same Entity. As noted earlier, the WebSocket System ensures that the entities' states are properly synchronised and transferred. Figure 6.5 presents a unified diagram depicting the data flow and sequence of operations, including all involved processes.

6.7.2 Spectral Denoising

Spectral denoising [101] involves decomposing images into a specific domain, typically frequency-related, where noise can be more easily distinguished from the true signal. In this domain, a thresholding function is applied, followed by an inverse transformation to produce a denoised image. The thresholding function operates in multiple dimensions, and a search algorithm is needed to identify effective coefficients. Using these

coefficients, we aim to assess their effectiveness in reducing noise in caustics within animations.

The image denoising is performed by a Python application called Spectral Image Denoising (SID), which applies a set of thresholding coefficients and configuration properties. To allow Forge to communicate with SID, a corresponding Proxy System was developed in Python. This allows the denoising logic to be imported directly into Forge without spawning additional processes. The system is triggered when an Entity contains both a `CoefficientConfig` molecule and a `Buffer` molecule (which holds the noisy and reference caustic buffers). Once these molecules are detected, denoising is performed, and the denoised caustics buffer is appended to the `Buffer` molecule.

The Spectral Evaluation System was designed to conduct this evaluation. Initially, it loads the configuration, which includes the coefficient values and the renderer settings for a 16-frame animation. Figure 6.6 illustrates the flow of data between systems and the component repository, as well as the operations performed by the System at each tick. At the start, the configuration required by both the Candela and SID systems is attached to a new Entity. As with the Renderer Evaluation System described in Section 6.7.1, the WebSocket System is launched in *server* mode, and both the Candela and SID systems run as separate processes, each wrapped by the WebSocket System in *client* mode.

During tick_1 , the Evaluation System remains idle, awaiting results. The Candela System (reused from Section 6.7.1) processes the `RenderConfig` molecule, renders the specified scene, and attaches the resulting frame buffers to the Entity. At this point, the `Buffer` molecule contains a noisy image with 16 spp and a reference image with 65536 spp. The SID System then processes these buffers, using the `CoefficientConfig` molecule and the noisy 16 spp image to perform denoising, appending the resulting denoised image to the `Buffer` molecule. The Metric System (also reused from Section 6.7.1) compares the noisy, reference, and denoised buffers, and attaches five metric results for both Noisy-Ref (noisy image compared to reference) and Denoised-Ref (denoised image compared to reference) comparisons using the `Metrics` molecule. In the next tick, the SID System gathers the metric results for the frame, continuing this cycle for all remaining frames.

Once all frames have been evaluated, tick_n initiates finalisation, cleans up by removing the Entity containing the attached molecules, and generates the result plot shown in Figure 6.7. The graph illustrates how Curvelet coefficients perform in denoising animations using the MSE metric on the same scene it was trained with, but viewed from different angles. The Denoised-Ref line consistently remains below the Noisy-Ref line, indicating that the denoising process was successful for this animation according to the MSE metric. Figure 6.8 displays the noisy and denoised buffers generated for Frame 8.

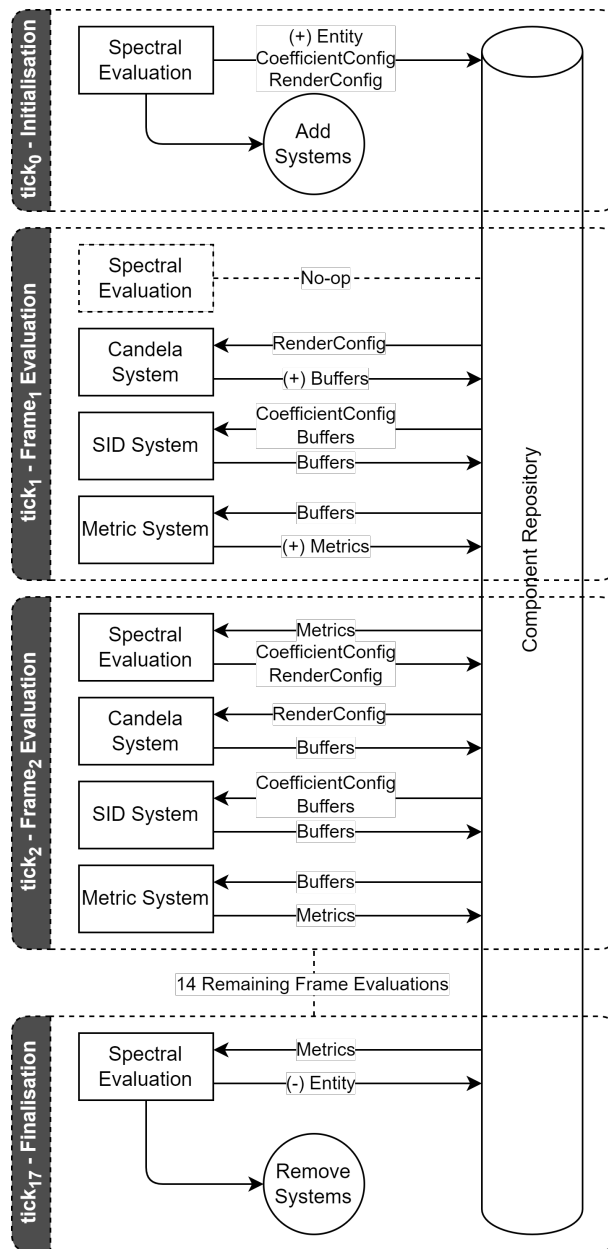


Figure 6.6 Spectral flow diagram.

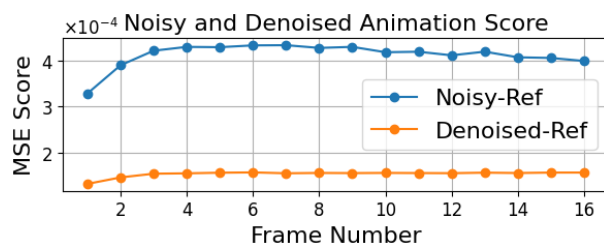


Figure 6.7 Spectral animation MSE score (lower is better).

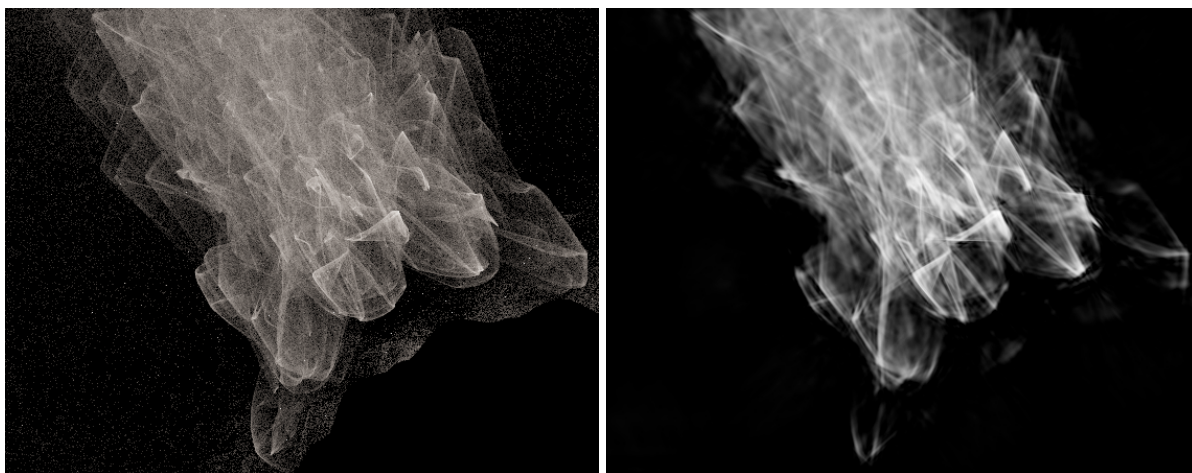


Figure 6.8 Frame 8. Left: noisy, right: denoised.

6.8 Discussion

Data in Anvil is organised into atoms and molecules, which are uniquely labelled data structures enriched with semantic meaning. This structure allows systems to operate on well-defined data. For example, if a system requires a `Metrics` component, it automatically knows that it will be working with a data structure containing five key image quality metrics: MSE, SSIM, FLIP, PSNR, and HDRVDP3. Anvil maintains a registry of components and systems, which facilitates the reuse of existing molecules when developing new systems, streamlining the development process.

The usability of Forge largely depends on the number of components, systems, and evaluation systems already available. For instance, the use case described in Section 6.7.2 leveraged both the Candela and Metric systems, significantly reducing the implementation effort. Additionally, Anvil, and by extension, Forge, is designed to be extensible, with a plug-in architecture that allows developers to easily create and share systems. It is also easy to see how confidence in the light and path tracing verification use case could be enhanced by incorporating other renderer systems that use `RenderConfig` molecules. For example, a Proxy System for the Mitsuba¹ renderer could be implemented, allowing verification against Mitsuba by simply registering its Proxy System during Forge’s initialisation phase.

Forge encourages developers to evaluate renderers without altering their code, thereby decoupling evaluation logic from application logic. As long as the renderer can be configured through the file system or an API, it can interface with Forge without requiring modifications. Forge also provides synchronisation mechanisms, such as the WebSocket System, which allow systems to run in isolated process spaces, including remote systems. This design promotes scalability, making it feasible for use in large-

¹<https://www.mitsuba-renderer.org/>

scale evaluations, such as those conducted in render farms.

Forge also supports real-time evaluation scenarios. For instance, a renderer may accept user input to change camera angles or adjust scene materials while it is running. The resulting frames can be processed by other systems to analyse aspects like temporal coherence, motion blur quality, or the effectiveness of different sampling strategies. This real-time feedback can be invaluable to artists and developers, enabling them to make informed decisions about scene composition, lighting, or algorithm selection during the content creation process. Additionally, real-time analysis can help identify bugs that might occur under rare or specific scene conditions. This dynamic flexibility illustrates Forge’s potential not only for post-render analysis but also as a powerful tool for interactive development and optimisation in computer graphics pipelines.

Forge’s architectural design not only makes evaluation more efficient but also minimises the potential for human error and configuration inconsistencies, which are frequent sources of inaccuracy in comparative CG studies. While Forge was originally designed for use in the context of CG, it can also be applied in other domains that have similar workflow characteristics.

6.9 Summary

In this chapter, we introduced Forge, an extension of Anvil designed to standardise and streamline evaluation in computer graphics pipelines. Forge addresses key challenges, including configuration consistency, human error reduction, and experimental reproducibility, providing a robust framework for accurate and reliable comparisons of rendering techniques.

Its modular architecture ensures flexibility, allowing seamless integration of new tools without disrupting existing workflows. The synchronisation interface enables decentralised, distributed operations, as demonstrated by the WebSocket system, which supports remote evaluations across different processes or machines. This enhances scalability, making Forge suitable for both small-scale experiments and large-scale rendering environments.

Case studies confirmed Forge’s effectiveness in verifying rendering techniques, measuring their performance, and evaluating new algorithms. By automating evaluations, it minimises human error and configuration mismatches, ensuring reliable results.

Developed in tandem with our research on spectral denoising for caustics, Forge proved essential in enabling cross-language interoperability (e.g., Python for spectral analysis, C++ for rendering) and distributed execution (handling software licensing constraints). It provides a unified platform for end-to-end evaluation, from noisy sig-

nal analysis to denoised output validation, ensuring methodological rigour in caustics denoising and beyond.

Forge is a versatile, ready-to-use framework that enhances research efficiency while maintaining accuracy, making it a valuable tool for both academic and industrial applications.

7 Spectral Denoising

The generation of photorealistic images through PBR often results in visual artefacts, notably caustics, which are complex light patterns created by the reflection or refraction of light off curved surfaces. The challenge of rendering such phenomena accurately at low sample counts is compounded by significant noise. This is clearly demonstrated in Chapter 5, where light tracing has been employed and demonstrated at 16 spp for the use in real-time rendering. Denoising is therefore a critical step in the rendering pipeline to enhance image quality with less computational costs than generating more samples.

Fourier, wavelet and curvelet transforms are traditional image transform techniques that have been employed extensively for noise reduction across various domains of digital image processing. These transforms decompose an image into components at different scales and orientations, allowing selective reconstruction that aims to retain important features while mitigating noise. However, the effectiveness of these techniques in denoising PBR-generated caustics at low sample counts remains unexplored, especially when compared to modern denoising algorithms developed specifically for graphics applications, such as NVIDIA’s OptiX AI-accelerated denoiser, ReBLUR and ReLAX.

Chapter 6 introduced Forge, an evaluation framework originally developed to facilitate the assessment of spectral denoising. However, mirroring the approach taken with Anvil, we identified the potential to generalise the framework, thereby extending its applicability to a wider range of problems within computer graphics and other domains. As Forge serves as a superset of Anvil, it retains the same operational structure for atoms and molecules, enabling the automatic reuse of existing hooks for real-time evaluation.

This chapter investigates the viability of traditional image transforms for denoising caustics in PBR scenarios and compares their performance against state-of-the-art graphics-specific denoisers. The quality of denoised images is quantitatively assessed using five metrics: Ψ LIP, HDRVDP3 (VDP3), MSE, PSNR and SSIM. These metrics facilitate a comprehensive evaluation by comparing the denoised images to a converged solution of the same scene, rendered with a high number of samples. Furthermore, to optimise the parameters of wavelet and curvelet transforms, a rigorous search using these metrics is conducted, supplemented by confidence interval testing to ensure robustness against potential local minima in parameter space.

7.1 Methodology

This study aims to investigate whether employing different transforms for denoising caustics could offer a viable alternative to the predominant state-of-the-art denoising methods commonly applied in real-time rendering. In this chapter we compare several algorithms using scenes featuring caustic phenomena.

Caustics represent a light phenomenon generated by rays following paths of the form $L(S|D)^*SDE$, as per Heckbert’s notation [102]. The appearance and characteristics of caustic patterns differ depending on the scene. Typically, caustics can appear smooth (low frequency) or exhibit significant variation (high frequency), featuring extensive curvature and intricate detail. Consequently, this research should encompass scenes showcasing various types of caustics. Specifically, our study focuses on caustics that concentrate light through transmission and display complex patterns.

The concept of utilising image transforms for caustics arises from the theory that it ought to be feasible to identify a basis that distinguishes caustics from the salt-and-pepper noise generated by Monte Carlo rendering techniques. As this noise is characterised by high frequency, employing image transforms like Fourier appears promising. However, since the Fourier transform acts globally on an image, influencing all areas, alternative transforms such as wavelets and curvelets should be considered. These additional transforms provide multi-resolution analysis and a basis specifically tailored for capturing curved structures.

Algorithm 4 Search

Require: Noisy image img_n ; ref. image img_r ; min. function M ; denoiser D ; initial coeffs. c_i ; thresholding function T ; search alg. S
Ensure: Optimal coeffs. c_o

- 1: $c_c \leftarrow c_i$
- 2: **while** $S.continue$ **do**
- 3: $c_c \leftarrow T(c_c)$
- 4: $img_d \leftarrow D(c_c, img_n)$
- 5: $score \leftarrow M(img_d, img_r)$
- 6: $c_c \leftarrow S.min(score)$
- 7: **end while**
- 8: $c_o \leftarrow c_c$

Algorithm 5 Comparator

Require: Run data r_1 ; Run data r_2

- 1: **for** $i \in [1, 2]$ **do**
- 2: **for all** $metric \in r_i$ **do**
- 3: $s_1 \leftarrow metric.noisyScore$
- 4: $s_2 \leftarrow metric.denoisedScore$
- 5: $d_i.append(s_1/s_2)$
- 6: **end for**
- 7: **end for**
- 8: $n_1, n_2 \leftarrow CountImprovedMetrics(d_1, d_2)$
- 9: **if** $n_1 \neq n_2$ **then return** $n_2 - n_1$
- 10: **end if**
- 11: $s_1, s_2 \leftarrow SumMetricRatios(d_1, d_2)$ **return** $s_2 - s_1$

At a high level, the denoising procedure through transforms entails a set sequence of steps. Initially, the noisy image undergoes transformation from the spatial to the transform domain. Subsequently, modifications are applied to the image in the transform domain before it is transformed back to the spatial domain, resulting in a denoised image. However, the challenge lies in determining how to enact changes in the transform domain to achieve image denoising. The transform domain may match or even exceed the size of the spatial domain, leading to a multitude of parameters that could be altered. Typically, conventional methods utilise thresholding functions

such as soft thresholding, which condense the variable parameters to a single value. Nonetheless, this approach severely restricts the degrees of freedom; altering just one parameter can have a widespread impact on features within the transform domain. To address this, partitioning the transform space is required. This involves defining distinct regions within the domain to group coefficients with similar properties, for instance by using frequency bands for the Fourier transform. The partitioning should yield sufficient parameters for denoising while avoiding an excessive number of them, which would render finding thresholding values impractical. Importantly, the partitioning strategy needs to be tailored to each transform, given that the characteristics of each transform domain differ.

To ascertain the efficacy of these image transforms in denoising, it is imperative to conduct an experiment. This experiment aims to determine suitable thresholding values, if applicable, for denoising. Identifying these values necessitates a search algorithm (Algorithm 4) capable of exploring the entire parameter search space. Various minimisation algorithms can be employed for this purpose. These algorithms require a function that, given a set of coefficients (representing the thresholding values), computes a score. This scoring function should accurately reflect the image quality, for which we utilise several algorithms commonly found in computer graphics literature. These algorithms receive the noisy image and the denoised image as input and provide a similarity value as output.

In addition to defining the function to be minimised, these search algorithms necessitate a description of the search space, including the minimum and maximum values for each variable, along with the initial starting point. Consequently, varying starting points may lead to different minimum outcomes. To ensure the discovery of a global or nearly global minimum, statistical analysis is conducted. Multiple runs are assessed with different randomly selected starting points to compute the standard error of the mean score. This process yields a confidence interval, indicating the probability that the true mean score lies within it. In our context, generating intervals spanning two standard deviations (95% confidence) should be adequate.

The experiment's outcomes are expected to yield the optimum denoised image for each scene. Nevertheless, it is imperative to determine the appropriate method for conducting the selection process. For each scene, a range of parameters is adjusted, leading to a minimisation process that generates a set of coefficients. Since multiple metrics are incorporated into the experiment, each scene will produce a top-ranking result for each metric. Selecting the best outcome should be approached objectively. One approach to achieving this is by selecting the result based on the number of metrics indicating improvement. If there are an equal number of improvements across metrics, the higher-scoring outcomes should be prioritised. Algorithm 5 depicts this process in more detail. The run data refers to the results from two searches. This data contains

the thresholding coefficients that were utilised to produce a denoised image, which are then used to calculate the image similarity score for each metric.

Ultimately, to assess whether image transforms offer a viable alternative to existing denoising methods, the same noisy images must undergo denoising using both approaches. Subsequently, the caustics buffer needs to be extracted post denoising. Following this, scores for each metric are generated, which are then compared to images denoised using image transforms. The determination of how many scenes outscore current denoisers will ascertain whether image transforms can indeed serve as a feasible alternative for denoising caustics.

To further assess the usability of the derived thresholding coefficients, additional experiments can be conducted. Specifically, it is important to determine whether these coefficients can be directly applied to denoise colour images. One approach involves processing each channel independently, treating them as individual greyscale luminance components. The effectiveness of this method can then be evaluated by comparing the denoised output with a reference colour image, measuring the improvement in noise reduction.

Another set of experiments should examine the robustness of the thresholding coefficients under various image transformations. This analysis will help determine whether the denoising process remains effective when applied to different views of the same scene, particularly when relying on a single image as the reference. Key transformations of interest include spatial translations and image flipping. The denoising performance can be quantified by comparing the processed output against the transformed reference using established image similarity metrics.

Finally, an investigation should be carried out to determine whether the thresholding coefficients exhibit consistency across different scenes. This would determine whether scenes with similar coefficient profiles share common characteristics such as high-frequency caustics. To facilitate this comparison, the cosine similarity metric can be employed, provided that the coefficient structures are compatible (i.e., they are derived from images of identical dimensions). Additionally, it would be valuable to assess whether these coefficients can be generalised, that is, whether they can be effectively interchanged between different scenes.

7.2 Experimental Setup

This section outlines the procedures implemented to establish the experiment. Section 7.2.1 outlines the selection criteria for scenes. Section 7.2.2 details the partitioning of transform domains for filtering purposes. Section 7.2.3 explains the process involved in conducting the search. Finally, Section 7.2.4 depicts the setup process for current de-

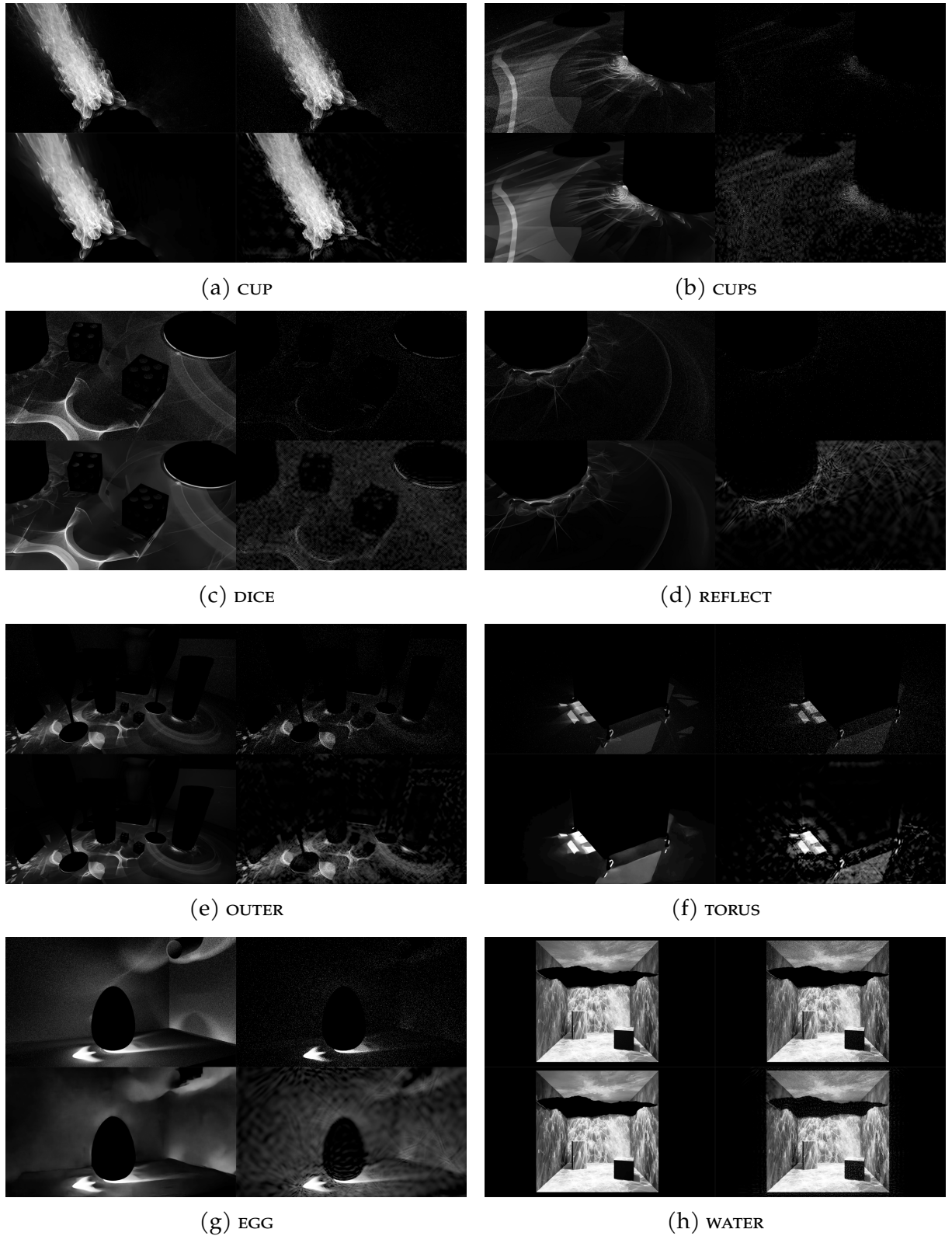


Figure 7.1 Collection of various images used in the study (image brightness enhanced for illustrative purposes). Each image is divided into four quadrants. Top-left: Ground truth image. Top-right: Noisy input. Bottom-left: Result from the OptiX denoiser. Bottom-right: Best denoised candidate from traditional denoisers.

noisers in the experiment.

7.2.1 Scene Preparation

Eight scenes were chosen (Figure 7.1), covering a range of frequencies, with some scenes having a mix of low and high frequencies. The scenes¹ include CUP (c), featuring caustics cast by a backlight onto a surface below a cup, and CUPS (s), DICE (d), REFLECT (r), and OUTER (o), showing different perspectives of the same scene. TORUS (t) showcases a torus within a glass cube, while EGG (e) features a glass egg on a table. Finally, WATER (w) presents the Cornell Box with a layer of water on top.

These scenes were produced using a path tracer and a light tracer, with radiance saved in Extended Dynamic Range (EXR) format with no loss to dynamic range. The light tracer was primarily employed for generating caustics. For each scene, a pair consisting of a noisy and a reference image was created. The noisy images were rendered at 16 spp, while the reference images were rendered at 65536 spp. All images have a resolution of 1600×900 . It is important to note that while the number of samples per pixel can be readily determined in a path tracer, this is not always straightforward in light tracing, as rays are traced from the light sources. Consequently, in the context of light tracing, the number of samples taken from the light sources corresponds to the total number of samples per frame in a path tracer.

7.2.2 Transform Partitioning

Fourier, Wavelet, and Curvelet are the primary transforms utilised. For wavelets, the symlets chosen include sym/2 and sym/20. The 2D Fourier transform produces an image in the frequency domain comprising complex numbers. This image can then be split into two parts, one depicting magnitude and the other depicting phase. In our experiments only the magnitude image is utilised, leaving the phase image unchanged when reverting to the spatial domain.

The magnitude image is divided in such a manner that the magnitude of the lowest frequency is situated at the centre. As one moves away from the centre, the frequency increases. Consequently, a circle centred on this image represents all pixels with the same frequency. Thus, two points on a circle denote a sinusoidal wave with the same frequency but varying orientation on the spatial image. It is noteworthy that if the original image is rectangular, the same frequencies are represented on an ellipse instead.

In this experiment we establish multiple frequency bands using ellipses, with each band assigned a threshold value, thereby creating a set of values that will filter the

¹<https://github.com/kvnnap/spectral-image-denoising>



Figure 7.2 Coefficient filters for DICE. Left: Fourier. Centre: Wavelet. Right: Curvelet.

ellipse region. This set can be modified, and an image can be generated by employing the inverse Fourier transform. Following this, the resulting image is assessed against a reference image using a metric, and a score is determined. The set is then adjusted until this score is minimised or maximised (depending on the selected metric). Figure 7.2 (left) demonstrates an example of such a filter.

The wavelet transform recursively breaks an image into progressively smaller scales, producing average and detail coefficients, which are split into horizontal, vertical, and diagonal components providing the DWT with directional sensitivity to eight orientations. The original image can then be reconstructed using the detail coefficients obtained at each scale.

In our experiments, we apply filtering to these detail coefficients (Figure 7.2, centre) with larger tiles representing higher frequency detail coefficients, while smaller tiles depict lower frequency bands. Filtering can be applied at a more precise level since each detail coefficient forms an image. However, we confine ourselves to filtering the entire coefficient image to maintain a manageable parameter space. No filtering is carried out on the average coefficient.

The curvelet transform breaks down an image into smaller scales, similar to wavelets, but each scale is further divided into multiple angles. This provides a higher level of directional sensitivity compared to wavelets. Figure 7.2 (right) illustrates these coefficients in polar coordinates. Similar to the Fast Fourier Transform (FFT), the low frequencies are positioned at the centre. Due to the significant number of coefficients at each scale, which increases with additional subdivisions, filtering is applied solely at the scale level. In the figure there are only seven scales, thus seven weights are adjusted accordingly.

7.2.3 Search

For every scene, the search is conducted using both the noisy image and the reference image. All images are caustic buffers produced from a light tracer. This buffer comprises light paths that meet the condition $L(S|D)*SDE$ (following Heckbert's notation [102]). Both the noisy and reference images are converted to greyscale and then

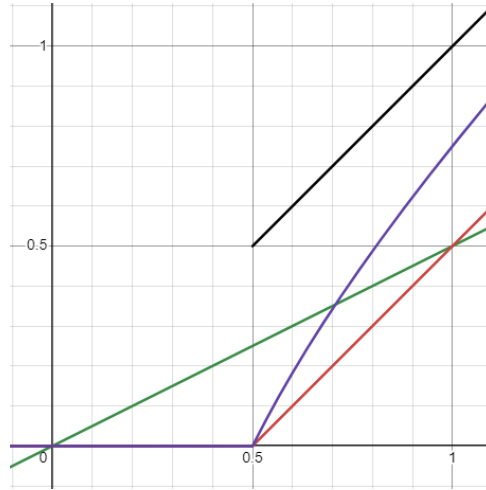


Figure 7.3 Thresholding functions ($t = 0.5$). Black is hard, red is soft, purple is garrote, green is multiplication.

undergo tone mapping using the Academy Colour Encoding System (ACES) before being transformed. Three transform algorithms were assessed: Fast Fourier Transform, Discrete Wavelet Transform, and Discrete Curvelet Transform. The transform domain is segmented (Section 7.2.2), with each segment assigned a thresholding coefficient. The range for each coefficient spans from zero to the maximum value of the segment when employing soft, hard, and garrote thresholding methods. The range for the multiplication threshold ranges from zero to one. The impact of thresholding with a value of 0.5 is illustrated in Figure 7.3.

An algorithm is employed to adjust the threshold coefficients allocated to each segment, and the inverse transform is calculated to derive the denoised image (in the spatial domain). The following search algorithms are utilised: Bayesian optimisation using Gaussian processes, Nelder-Mead, L-BFGS-B, TNC, SLSQP, Powell, trust-constr, and COBYLA. Subsequently, the denoised image is compared to the reference image using SSIM, MSE, PSNR, HDRVDP3 [38], or FLIP metrics. FLIP and HDRVDP3 are set with a pixel-per-degree measurement of 20.4, which corresponds to an observer viewing a 24-inch Full High Definition (FHD) display from a distance of 30 cm. This process is reiterated until the score is minimised/maximised or until a predetermined maximum number of iterations is attained (100). Details regarding the filtering procedure for each image transformation algorithm can be found in Section 7.2.2.

7.2.4 Denoisers

Three denoisers, ReLAX, ReBLUR, and OptiX, are employed to produce denoised caustic buffers. In the cases of ReLAX and ReBLUR, the noisy caustic signal is integrated into the diffuse chain alongside other essential buffers, including motion vectors, normals and roughness, z-buffer, and ray-hit distance. Two experiments are conducted:

one where the input caustic signal accumulates over 16 frames, equivalent to 16 spp, and another where it is reset per frame. Both ReLAX and ReBLUR serve as temporal denoisers, and it is anticipated that their results will be similar for a small accumulation. Nonetheless, the denoised image represents the equivalent of 16 spp. For OptiX, only the caustic signal is incorporated and is consistently accumulated up to 16 spp. For each scene, the denoised signal is assessed against the reference signal using the aforementioned five metrics, and the similarity score is recorded.

7.2.5 Confidence

The thresholding coefficients are defined within a specific range, which the search algorithm explores to determine the final minimisation point. In our experiments, the search is consistently initialised at the midpoint of this range. However, this approach may lead to convergence at a local minimum, potentially suboptimal for the overall solution.

To enhance the robustness of our results, we conduct multiple runs of the same search algorithm, utilising identical metrics, denoising methods, and thresholding functions, but with randomised initialisation points within the coefficient range. Specifically, we perform 100 independent runs to ensure statistical reliability. The standard error of the mean is then computed, and a 95% confidence interval is derived (spanning two standard deviations), providing a high probability that the true mean lies within this range. We then compare whether the original run (starting at the midpoint) lies within this interval.

Given the computational intensity of this process, we restrict our confidence interval analysis to the aggregate metric scores, ensuring a balance between statistical rigour and computational feasibility.

7.2.6 Colour

This experiment investigates whether thresholding coefficients derived from greyscale images can be effectively applied to denoise colour images. For each scene we apply these coefficients independently to each channel of its noisy colour counterpart. The denoised output is then compared both to the original noisy image and the reference colour image to assess any visual and quantitative improvements. Image similarity metrics are also applied and results tabulated in order to measure the performance objectively.

7.2.7 Image Transformation

In this part of the experiment, the aim is to examine whether image transformations, specifically translations and flips, applied to noisy images influence the denoising performance when the same set of coefficients is used for reconstruction. As in previous experiments, a high-performing scene is selected, and the image is subjected to eight incremental left-to-right translations and eight incremental top-to-bottom translations. Each transformation is applied to three variants of the image: the original (unaltered), the horizontally flipped version, and the vertically flipped version.

For each transformation, image similarity scores are computed for both the noisy and denoised images. These results are visualised in a plot, allowing for an assessment of whether the denoised images consistently, partially, or never outperform their noisy counterparts in terms of similarity. This analysis provides insight into the robustness of the denoising process under spatial transformations.

7.2.8 Coefficient Scene Similarity

The objective of this experiment is to investigate whether different scenes produce similar thresholding coefficients. To this end, cosine similarity is employed as a means of comparing the thresholding coefficients. The experimental procedure involves loading all available runs, including the best-performing candidates identified during the search process, and computing cosine similarity between runs that utilise different scenes but share the same spectral denoising algorithm and thresholding method.

This restriction is imposed for two reasons. Firstly, different spectral denoising algorithms impose distinct structural characteristics on the resulting coefficients, rendering direct comparison impossible. Secondly, although the structural format of the thresholding coefficients remains consistent across different thresholding functions, their semantic interpretation is function-dependent and therefore not directly comparable.

A similarity matrix is constructed, enabling similarity queries to be performed. These queries facilitate the identification of runs that are most similar to a given run within the current working set. Quantitative analysis is then conducted to explore the similarity between scenes.

7.2.9 Different Scenes

A final experiment is conducted to evaluate whether thresholding coefficients derived from one scene can be effectively applied to denoise other scenes, possibly guided by the results obtained from the cosine similarity process. A key constraint in this inves-

tigation is that all images must share identical dimensions to ensure comparability. In each trial, a target scene is selected and denoising is carried out using thresholding coefficients obtained through searches conducted on different source scenes.

The resulting noisy and denoised images are then compared to the corresponding reference images using established image similarity metrics. The effectiveness of the denoising process is assessed based on the similarity scores obtained, providing an indication of how well the transferred coefficients generalise across scenes.

7.3 Evaluation

Table 7.1 presents the scores for each metric on each noisy image. Table 7.2 lists the ReLAX and ReBLUR scores for denoised images with and without caustic buffer accumulation. Table 7.3 details the parameters used to achieve the best scores across all metrics. Table 7.4 shows the best aggregate scores sorted using Algorithm 5, while Table 7.5 lists the parameters used to attain these scores. Table 7.6 contains the scores for OptiX with caustic accumulation. Table 7.7 summarises the best scores across all metrics. Table 7.8 compares the top image transform denoisers (curvelet-based) with the OptiX denoiser.

	$\mathbb{H}LIP$	VDP3	MSE	PSNR	SSIM
C	4.95e-02	4.77	1.46e-03	28.35	0.76
S	9.99e-02	3.04	9.96e-03	20.02	0.45
D	1.08e-01	3.33	1.33e-02	18.78	0.35
R	2.87e-02	2.96	2.41e-03	26.18	0.66
O	5.76e-02	2.34	5.16e-03	22.88	0.63
T	1.44e-02	3.74	6.92e-04	31.60	0.85
E	8.67e-02	1.43	9.25e-03	20.34	0.37
W	7.51e-02	6.03	1.66e-03	27.71	0.83

Table 7.1 Metric scores for the noisy images.

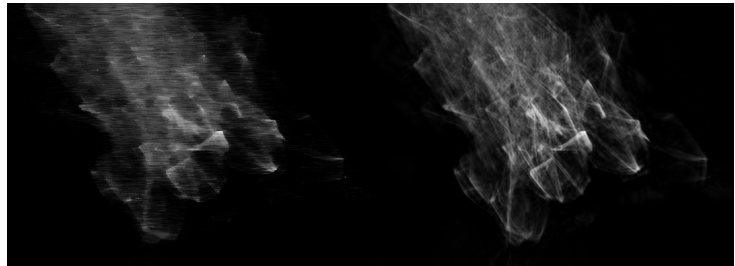


Figure 7.4 Left: ReLAX denoised. Right: Curvelet denoised.

When comparing the best results in Table 7.7 with those of ReLAX in Table 7.2, it is evident that CUP, REFLECT, OUTER, EGG and WATER performed better across all metrics. CUPS and DICE performed better in HDRVDP3, PSNR and SSIM. TORUS performed well in HDRVDP3 and MSE. Compared with ReBLUR, all scenes demonstrated enhanced performance across all metrics, except DICE, which had worse MSE scores. Compared to OptiX, most results were generally inferior. However, HDRVDP3 scores improved across all scenes except WATER. CUPS, DICE, REFLECT and EGG had better PSNR and SSIM scores. Additionally, TORUS performed better in PSNR and REFLECT in $\mathbb{H}LIP$. Table 7.3 shows that the curvelet transform generally performed well as it is used in 25 times out of the 40 best scorers (62.5%). Additionally, Powell performed well across search algorithms as it is used 30 times out of the 40 best scorers (75%). Table 7.8 shows that

	FLIP	FLIP	VDP3	VDP3	MSE	MSE	PSNR	PSNR	SSIM	SSIM
	Acc.		Acc.		Acc.		Acc.		Acc.	
C	2.79e-02	2.68e-02	7.41	7.46	3.88e-04	3.88e-04	33.79	33.79	0.92	0.92
S	8.17e-02	8.34e-02	2.59	2.31	4.81e-03	5.19e-03	23.18	22.85	0.52	0.51
D	8.32e-02	8.39e-02	1.85	1.64	5.57e-03	5.67e-03	22.54	22.22	0.42	0.39
R	2.27e-02	2.25e-02	4.96	4.74	6.53e-04	6.05e-04	31.85	32.18	0.72	0.72
O	2.81e-02	2.83e-02	5.75	5.65	6.70e-04	7.30e-04	31.74	31.28	0.83	0.83
T	5.49e-03	5.23e-03	8.17	8.06	3.20e-05	3.46e-05	43.90	37.03	0.97	0.95
E	5.26e-02	5.29e-02	4.45	4.33	1.34e-03	1.40e-03	28.71	28.55	0.56	0.57
W	7.55e-02	8.33e-02	5.18	4.45	1.95e-03	2.36e-03	26.97	26.15	0.83	0.80
C	2.98e-02	2.87e-02	6.72	7.51	4.85e-04	4.15e-04	32.82	33.49	0.91	0.92
S	8.52e-02	8.63e-02	2.27	1.23	5.30e-03	5.95e-03	22.76	22.26	0.50	0.49
D	8.81e-02	8.89e-02	2.36	1.12	5.98e-03	7.46e-03	21.98	21.02	0.39	0.37
R	2.33e-02	2.20e-02	4.98	3.69	5.71e-04	4.73e-04	31.84	32.52	0.70	0.69
O	3.15e-02	3.45e-02	5.44	4.72	9.12e-04	1.20e-03	30.32	29.13	0.81	0.77
T	6.39e-03	7.28e-03	8.08	7.82	4.69e-05	7.13e-05	35.19	33.36	0.93	0.92
E	5.44e-02	5.76e-02	4.51	3.69	1.68e-03	1.94e-03	27.76	27.13	0.55	0.52
W	8.07e-02	9.09e-02	4.59	4.73	2.27e-03	2.72e-03	26.30	25.52	0.82	0.79

Table 7.2 Accumulating and non-accumulating scores. Best score for each scene-metric pair shown in bold. Top: ReLAX. Bottom: ReBLUR.

	Metric	Search	Denoiser	Thres.	$T_s(s)$	$T_d(s)$	Score
C	FLIP	powell	curvelet	soft	1511	3.30	2.32e-02
C	VDP3	powell	curvelet	soft	2978	3.17	8.51
C	MSE	powell	curvelet	garrote	496	3.16	1.87e-04
C	PSNR	trust	curvelet	garrote	1486	3.27	36.95
C	SSIM	powell	curvelet	soft	878	3.27	0.95
S	FLIP	powell	curvelet	soft	649	3.30	8.36e-02
S	VDP3	cobyla	fourier	mult	1705	0.28	4.54
S	MSE	cobyla	fourier	mult	91	0.30	5.08e-03
S	PSNR	powell	wav/20	hard	97	0.19	26.98
S	SSIM	powell	wav/20	hard	190	0.21	0.65
D	FLIP	powell	curvelet	garrote	3.71	638	8.53e-02
D	VDP3	cobyla	fourier	mult	1705	0.29	4.53
D	MSE	cobyla	fourier	mult	107	0.30	6.55e-03
D	PSNR	cobyla	curvelet	soft	436	3.08	28.15
D	SSIM	powell	wavelet	hard	555	0.05	0.55
R	FLIP	powell	curvelet	soft	899	3.13	2.15e-02
R	VDP3	cobyla	fourier	mult	1634	0.26	6.18
R	MSE	powell	curvelet	hard	782	3.08	3.80e-04
R	PSNR	powell	wavelet	hard	153	0.05	34.60
R	SSIM	powell	wavelet	hard	364	0.05	0.81
O	FLIP	powell	curvelet	soft	865	3.09	2.80e-02
O	VDP3	powell	curvelet	hard	6220	2.97	7.14
O	MSE	powell	curvelet	garrote	752	3.00	4.41e-04
O	PSNR	powell	wavelet	hard	190	0.05	33.98
O	SSIM	powell	curvelet	mult	1270	3.06	0.88
T	FLIP	powell	curvelet	soft	1003	2.35	6.31e-03
T	VDP3	powell	curvelet	soft	2698	2.20	8.39
T	MSE	powell	curvelet	soft	694	2.21	2.47e-05
T	PSNR	powell	wavelet	hard	80	0.02	39.96
T	SSIM	powell	curvelet	hard	734	2.18	0.95
E	FLIP	powell	curvelet	soft	1895	3.12	5.05e-02
E	VDP3	powell	fourier	garrote	21164	0.29	5.71
E	MSE	powell	curvelet	hard	632	3.13	1.06e-03
E	PSNR	gp	wavelet	hard	577	0.05	34.09
E	SSIM	powell	wavelet	garrote	446	0.05	0.85
W	FLIP	powell	curvelet	garrote	814	1.34	5.40e-02
W	VDP3	powell	curvelet	garrote	1976	1.35	7.90
W	MSE	trust	curvelet	garrote	748	1.39	4.91e-04
W	PSNR	bfgs	curvelet	garrote	1241	1.33	32.96
W	SSIM	powell	curvelet	garrote	1832	1.33	0.92

Table 7.3 Best scores from multiple runs (detailed); gp is Bayesian optimisation. T_s is search time, T_d is denoising time.²

	FLIP	VDP3	MSE	PSNR	SSIM		Metric	Search	Denoiser	Thres.	T _s (s)	T _d (s)	Score
C	2.39e-02	8.34	1.87e-04	36.95	0.94	C	MSE	trust	curvelet	garrote	722	3.26	1.87e-04
S	8.86e-02	3.18	5.35e-03	26.98	0.65	S	PSNR	powell	wav/20	hard	97	0.18	26.98
D	9.62e-02	3.41	6.88e-03	23.38	0.51	D	PSNR	cobyla	wav/20	hard	33	0.17	23.38
R	2.58e-02	5.34	3.80e-04	33.47	0.77	R	PSNR	powell	curvelet	hard	546	3.25	33.47
O	2.98e-02	6.79	4.41e-04	33.48	0.85	O	MSE	trust	curvelet	garrote	881	3.11	4.41e-04
T	6.77e-03	8.32	2.47e-05	37.97	0.93	T	MSE	trust	curvelet	soft	577	2.28	2.47e-05
E	5.46e-02	4.09	1.06e-03	30.41	0.74	E	MSE	powell	curvelet	garrote	381	3.21	1.06e-03
W	5.28e-02	7.89	4.91e-04	32.96	0.92	W	MSE	bfgs	curvelet	garrote	697	1.35	4.91e-04

Table 7.4 Aggregate metrics with all metrics computed.

Table 7.5 Aggregate metrics. T_s is search time, T_d is denoising time.²

the AI-based OptiX denoiser performed worse only in `REFLECT` when measured with the `FLIP` metric against the curvelet-based denoiser.

In the aggregate metrics presented in Table 7.5, MSE and PSNR were the most effective performance measures. The optimisation methods were distributed as follows: the Powell and trust-constr algorithms were employed equally (three times each), while the COBYLA and BFGS methods were each used once. Among the thresholding functions, the garrote approach was the most frequently applied, followed by hard and soft thresholding.

Figure 7.5 illustrates the denoising performance of the `DICE` metric, comparing the results obtained using the ReLAX, ReBLUR and Optix methods with those of conventional denoising techniques, specifically the curvelet and wavelet (Symlet-20) transforms.

7.3.1 Confidence Interval

Table 7.9 presents the confidence intervals for the aggregated metrics. Here, $\text{Score}_{\text{mid}}$ denotes the score obtained when the initial search point is set to the midpoint of the input range, while $\text{Score}_{\text{min}}$ and $\text{Score}_{\text{max}}$ define the 95% confidence interval for the true mean. These scores were derived from 100 searches consisting of 100 maximum iterations each, where the initial search point was randomly selected within the valid input range.

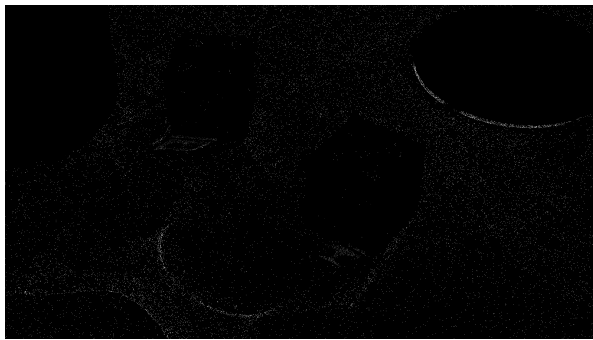
The results indicate that $\text{Score}_{\text{mid}}$ falls within the confidence interval in three out of eight cases, suggesting alignment with the expected mean. In the remaining five instances, $\text{Score}_{\text{mid}}$ lies outside the interval but consistently outperforms the mean, indicating statistically superior performance in these scenarios.

7.3.2 Colour Images

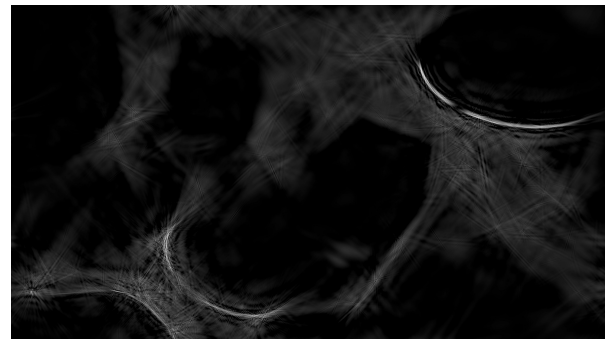
Table 7.10 presents the metric scores computed from the noisy RGB inputs. This table is analogous to Table 7.1, but with metrics evaluated across all three colour channels. Table 7.11 (similar to Table 7.7) reports the corresponding scores for the denoised RGB images, where the thresholding coefficients were derived from their greyscale counterparts. Finally, Table 7.12 (similar to Table 7.4) summarises the scores for the scenes that achieved the highest aggregate performance in prior evaluations, again using thresholding coefficients derived from their greyscale counterparts.

The results in both Table 7.11 and Table 7.12 outperform the noisy counterparts depicted in Table 7.10 across all metrics. Figure 7.6 illustrates the caustic buffers com-

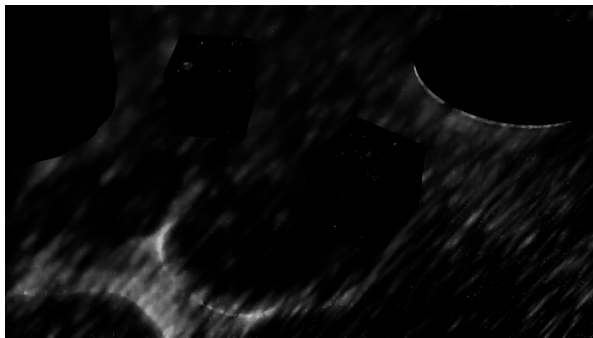
²Search runs distributed on AMD Ryzen 3600X, 3960X, 7950X, Intel i7-8700K (a) + NVIDIA RTX 2080 Ti (b). Only `FLIP` on (b). Denoising on (a).



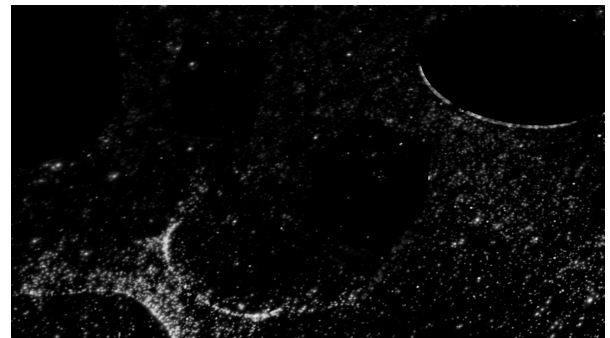
(a) NOISY



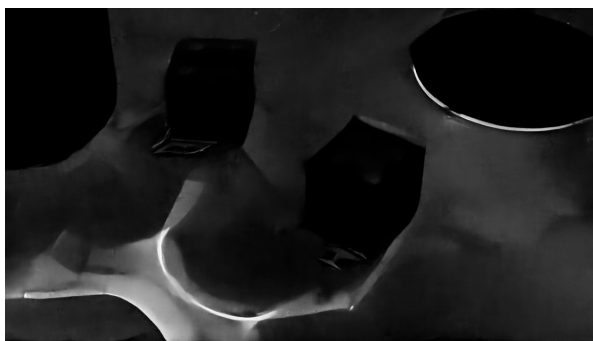
(b) FLIP (CURVELET)



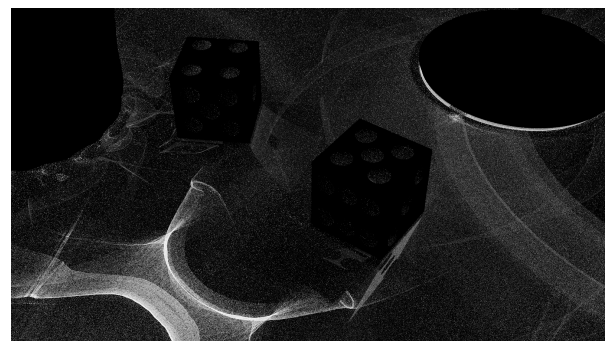
(c) RELAX



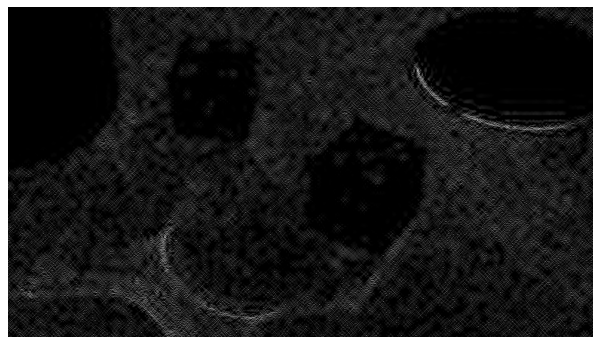
(d) REBLUR



(e) OPTIX



(f) REFERENCE



(g) AGGREGATE (PSNR Wav/20)

Figure 7.5 Dice scene denoised

	$\mathbb{F}$ LIP	VDP3	MSE	PSNR	SSIM		$\mathbb{F}$ LIP	VDP3	MSE	PSNR	SSIM
C	1.98e-02	7.75	1.56e-04	37.75	0.96	C	2.32e-02	8.51	1.87e-04	36.95	0.95
S	8.09e-02	2.58	2.58e-03	25.88	0.63	S	8.36e-02	4.54	5.08e-03	26.98	0.65
D	7.94e-02	1.94	2.93e-03	25.08	0.53	D	8.53e-02	4.53	6.55e-03	28.15	0.55
R	2.65e-02	5.31	3.28e-04	34.11	0.79	R	2.15e-02	6.18	3.80e-04	34.60	0.81
O	2.33e-02	6.44	2.90e-04	35.30	0.90	O	2.80e-02	7.14	4.41e-04	33.98	0.88
T	4.36e-03	8.32	2.01e-05	38.85	0.96	T	6.31e-03	8.39	2.47e-05	39.96	0.95
E	4.45e-02	4.54	5.44e-04	32.65	0.77	E	5.05e-02	5.71	1.06e-03	34.09	0.85
W	4.12e-02	7.93	3.31e-04	34.67	0.95	W	5.40e-02	7.90	4.91e-04	32.96	0.92

Table 7.6 OptiX accumulating scores.

Table 7.7 Best scores from multiple runs.

	$\mathbb{F}$ LIP _{crv}	$\mathbb{F}$ LIP _{optix}
C	2.32e-02	1.98e-02
S	8.36e-02	8.09e-02
D	8.53e-02	7.94e-02
R	2.15e-02	2.65e-02
O	2.80e-02	2.33e-02
T	6.31e-02	4.36e-03
E	5.05e-02	4.45e-02
W	5.40e-02	4.12e-02

Table 7.8 $\mathbb{F}$ LIP metric comparison between the best curvelet-based denoiser and the OptiX denoiser. Figures in bold indicate the instances where the curvelet-based denoiser obtained a better metric value.

	Score _{mid}	Score _{min}	Score _{max}	Exponent
C	1.874103	1.868048	1.912491	e-04
S	26.97636	26.49668	26.79328	e-00
D	23.37665	22.76890	22.98302	e-00
R	33.46719	33.44750	33.45893	e-00
O	4.407896	4.353467	4.704791	e-04
T	2.468524	2.475348	2.478926	e-05
E	1.059136	1.059075	1.059324	e-03
W	4.910433	5.316091	5.967164	e-04

Table 7.9 Aggregate metrics 95% confidence interval. Values in bold represent scores outside the interval.

puted using the greyscale thresholding coefficients of the highest aggregate scores.

7.3.3 Image Transformation

Figure 7.7 displays the metric ratio analysis for translated variants of OUTER, employing thresholding coefficients obtained from the optimal aggregate score configuration. The figure comprises five plots, each representing a different evaluation metric. The x-axis denotes the translation increment applied to the image (measured in pixel multiples), with the original 1600×900 pixel image undergoing ten incremental translations. Each

	FLIP	VDP3	MSE	PSNR	SSIM
C	5.09e-02	4.28	1.45e-03	28.38	0.76
S	1.00e-01	3.50	9.95e-03	20.02	0.45
D	1.14e-01	3.54	1.32e-02	18.79	0.36
R	3.06e-02	3.97	2.36e-03	26.27	0.67
O	5.95e-02	3.07	5.01e-03	23.01	0.64
T	1.53e-02	3.65	6.37e-04	31.96	0.86
E	9.34e-02	2.25	8.72e-03	20.59	0.40
W	9.27e-02	6.09	1.73e-03	27.62	0.84

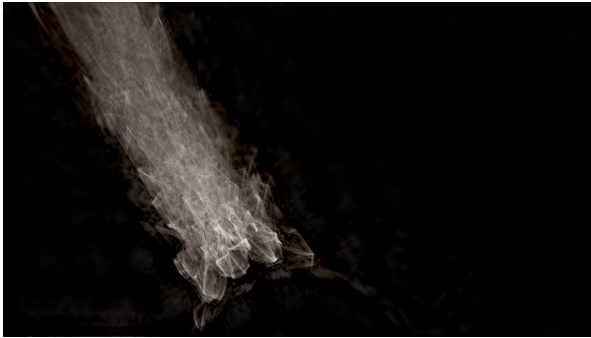
Table 7.10 Metric scores for the noisy coloured images.

	FLIP	VDP3	MSE	PSNR	SSIM
C	2.42e-02	8.16	1.87e-04	37.03	0.95
S	8.37e-02	5.01	5.08e-03	26.98	0.65
D	9.00e-02	4.90	6.77e-03	28.40	0.55
R	2.31e-02	6.38	4.39e-04	34.44	0.82
O	2.85e-02	6.74	4.46e-04	33.88	0.87
T	6.51e-03	7.93	2.33e-05	40.12	0.96
E	5.52e-02	5.84	1.12e-03	33.84	0.85
W	6.50e-02	7.93	4.89e-04	32.99	0.92

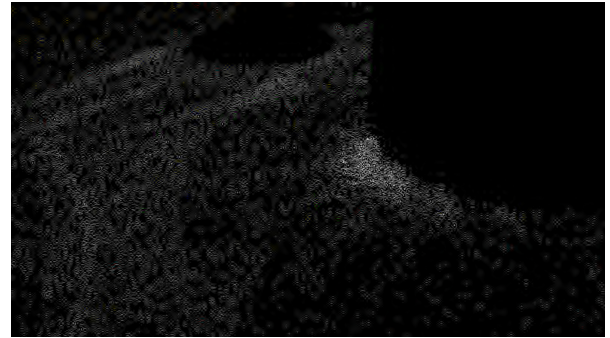
Table 7.11 Metric scores for the denoised coloured images.

	FLIP	VDP3	MSE	PSNR	SSIM	Denoiser	T_d(s)
C	2.49e-02	8.08	1.87e-04	37.03	0.94	curvelet	10.53
S	8.87e-02	3.89	5.35e-03	26.98	0.65	wav/20	0.65
D	9.98e-02	4.33	7.09e-03	23.25	0.51	wav/20	0.60
R	2.68e-02	4.81	4.39e-04	33.35	0.79	curvelet	10.77
O	3.01e-02	6.30	4.47e-04	33.44	0.86	curvelet	10.68
T	6.96e-03	7.81	2.34e-05	39.68	0.95	curvelet	10.62
E	5.85e-02	4.02	1.13e-03	30.14	0.75	curvelet	10.81
W	6.37e-02	7.92	4.89e-04	32.99	0.92	curvelet	10.47

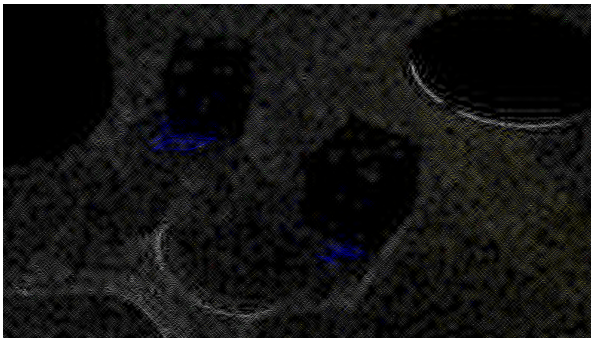
Table 7.12 Aggregate scores for the denoised coloured images, including denoising time.



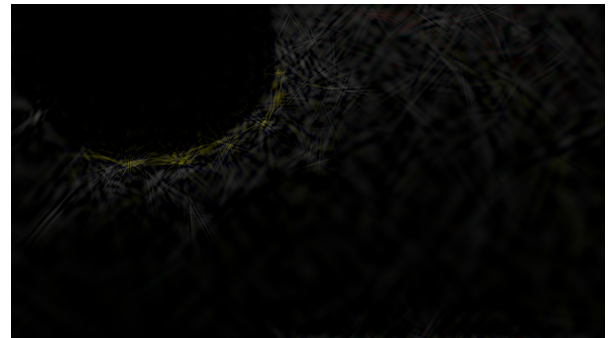
(a) CUP



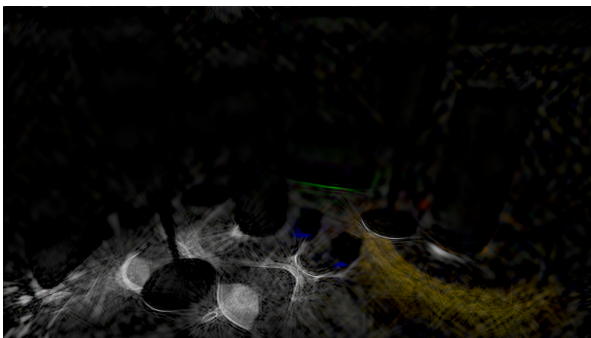
(b) CUPS



(c) DICE



(d) REFLECT



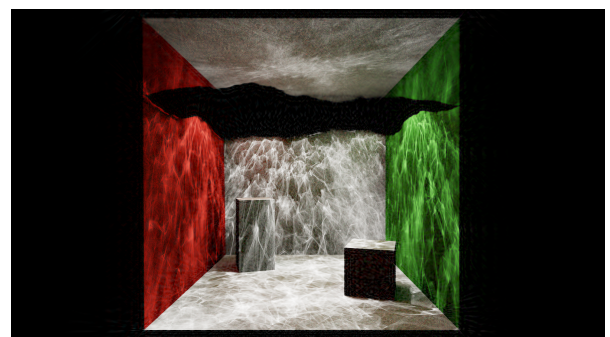
(e) OUTER



(f) TORUS



(g) EGG



(h) WATER

Figure 7.6 RGB denoised

increment corresponds to a horizontal shift of 160 pixels along the x-axis and a vertical shift of 90 pixels along the y-axis. For the identity transformation, translations are applied as described, whereas for the *FlipLR* and *FlipUP* operations, the source images undergo horizontal and vertical flipping respectively before translation.

A zero increment corresponds to no translation being applied, while an increment of ten is equivalent to zero due to the cyclical boundary conditions. As illustrated in the graph, a zero increment results in identical starting points for all horizontal and vertical translations across each transformation subgroup - namely, the identity transformation, *FlipLR* (horizontal flip), and *FlipUP* (vertical flip) since precisely no spatial translation has been performed at this stage. Additionally, the ratio observed at increment zero for the identity transformation (blue) corresponds to the ratio obtained by dividing the OUTER row values in Table 7.4 by those in Table 7.1 (or vice versa, see Equation 7.1) for the respective image quality metric.

The y-axis presents the metric ratio, which assesses performance relative to the noisy reference image. The metric ratio is defined such that:

- A value of unity (1) indicates parity between denoised and noisy images
- Values below 1 signify degradation in the denoised output
- Values above 1 demonstrate improvement, with a ratio of 2, for instance, denoting a twofold enhancement

The calculation of this ratio follows Equation 7.1, which provides the formal mathematical definition.

$$ratio = \begin{cases} \frac{noisy}{denoised} & \text{for MSE and FLIP (lower is better),} \\ \frac{denoised}{noisy} & \text{for SSIM, PSNR, and VDP3 (higher is better).} \end{cases} \quad (7.1)$$

For all metrics, it can be observed that all computed ratios exceed unity, indicating that every translation applied effectively performed denoising. In this implementation, denoising was achieved using curvelet transformation with Garrote thresholding. Furthermore, the improvement ratios along each curve in the plots demonstrate minimal variation when compared to the zero-increment case. For practical purposes, these ratios may be considered constant.

7.3.4 Coefficient Scene Similarity

Table 7.13 presents the cosine similarity values computed across all scene combinations. The majority of values register as zero, indicating either structural incompatibility between coefficient sets (arising from different denoisers) or semantic incompatibility due

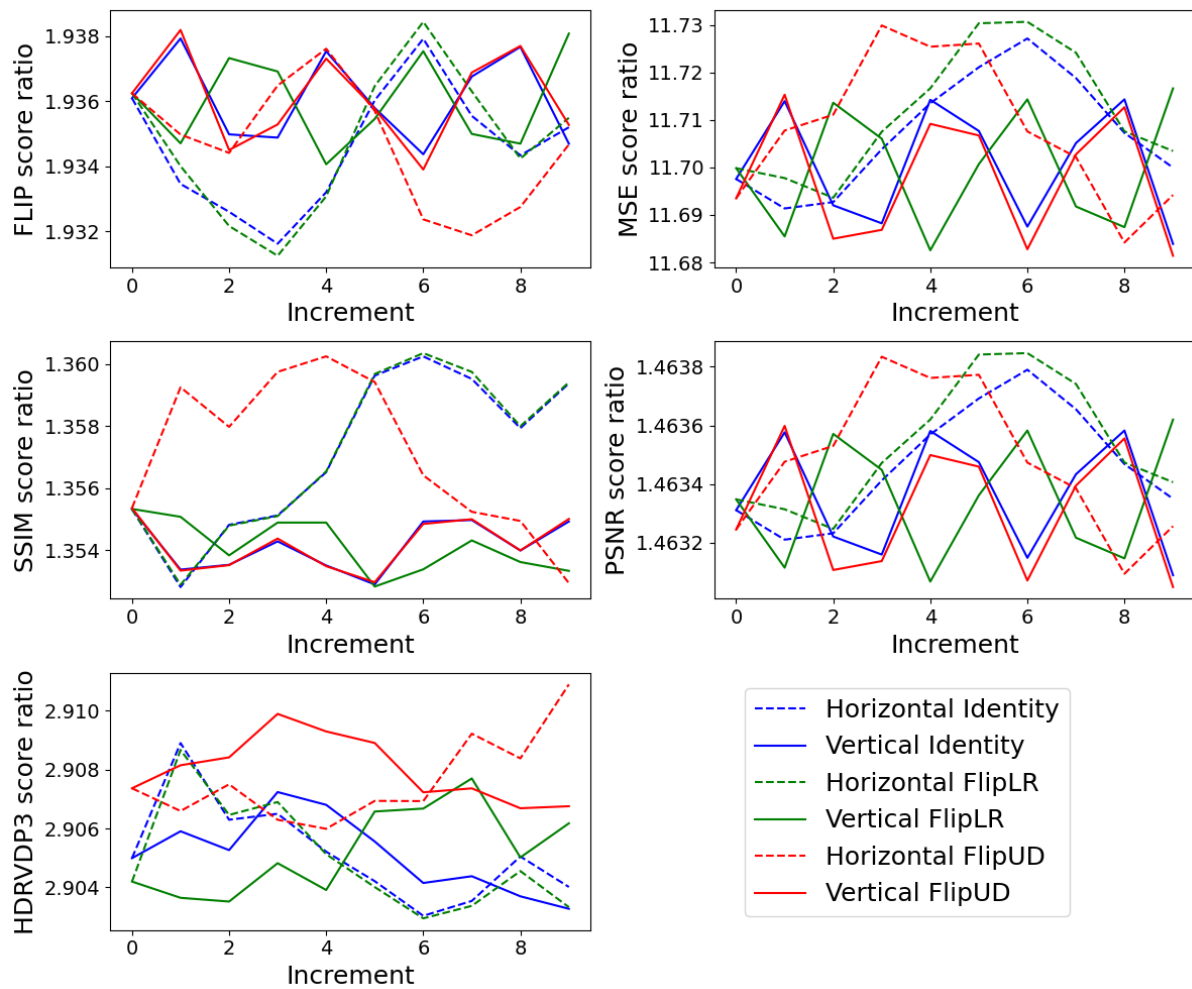


Figure 7.7 Image translations applied to OUTER (aggregate coefficients)

to distinct thresholding methodologies. Notably, the highest similarity scores emerge between WATER and EGG (0.99), followed by CUP and OUTER (0.98), and CUPS and DICE (0.85). Moderate similarity is observed between OUTER and WATER (0.37) and OUTER and EGG (0.33). The lowest similarity metrics occur between CUP and EGG (0.12) and CUP and WATER (0.16), preceding the zero-valued incompatibilities.

	C	S	D	R	O	T	E	W
C	0.00	0.00	0.00	0.00	0.98	0.00	0.12	0.16
S	0.00	0.00	0.85	0.00	0.00	0.00	0.00	0.00
D	0.00	0.85	0.00	0.00	0.00	0.00	0.00	0.00
R	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
O	0.98	0.00	0.00	0.00	0.00	0.00	0.33	0.37
T	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
E	0.12	0.00	0.00	0.00	0.33	0.00	0.00	0.99
W	0.16	0.00	0.00	0.00	0.37	0.00	0.99	0.00

Table 7.13 Aggregate scores cosine similarity matrix.

7.3.5 Different Scenes

Building upon the cosine similarity values, we apply denoising to four scenes using coefficients obtained from searches based on other scenes. Furthermore, to assess whether the similarity values in Table 7.13 correspond to any discernible visual patterns, we select two similar and two dissimilar cases for analysis. Figure 7.8 presents the four denoised images. Specifically, Figure 7.8a displays the denoised CUP image using thresholding coefficients derived from EGG, Figure 7.8b shows the denoised CUP image with coefficients from OUTER, Figure 7.8c illustrates the denoised EGG image using coefficients from CUP, and Figure 7.8d depicts the denoised OUTER image with coefficients derived from CUP.

Table 7.14 presents the image similarity metrics computed for the denoised images. When compared to the noisy values in Table 7.1, all evaluated scenes exhibit higher scores across all metrics, indicating that the denoising process is measurably effective. However, when benchmarked against the aggregate metrics of the denoised images in Table 7.4, these results outperform it only in the case of the OUTER image denoised using the CUP reference, and solely under the VDP3 metric.

7.3.6 Discussion

In the aforementioned results concerning denoising efficacy, spectral transform denoisers surpassed ReLAX at equivalent noise levels. For instance, in the CUP (Figure 7.4) and DICE (Figure 7.5) scenes, caustic patterns were rendered with greater definition when

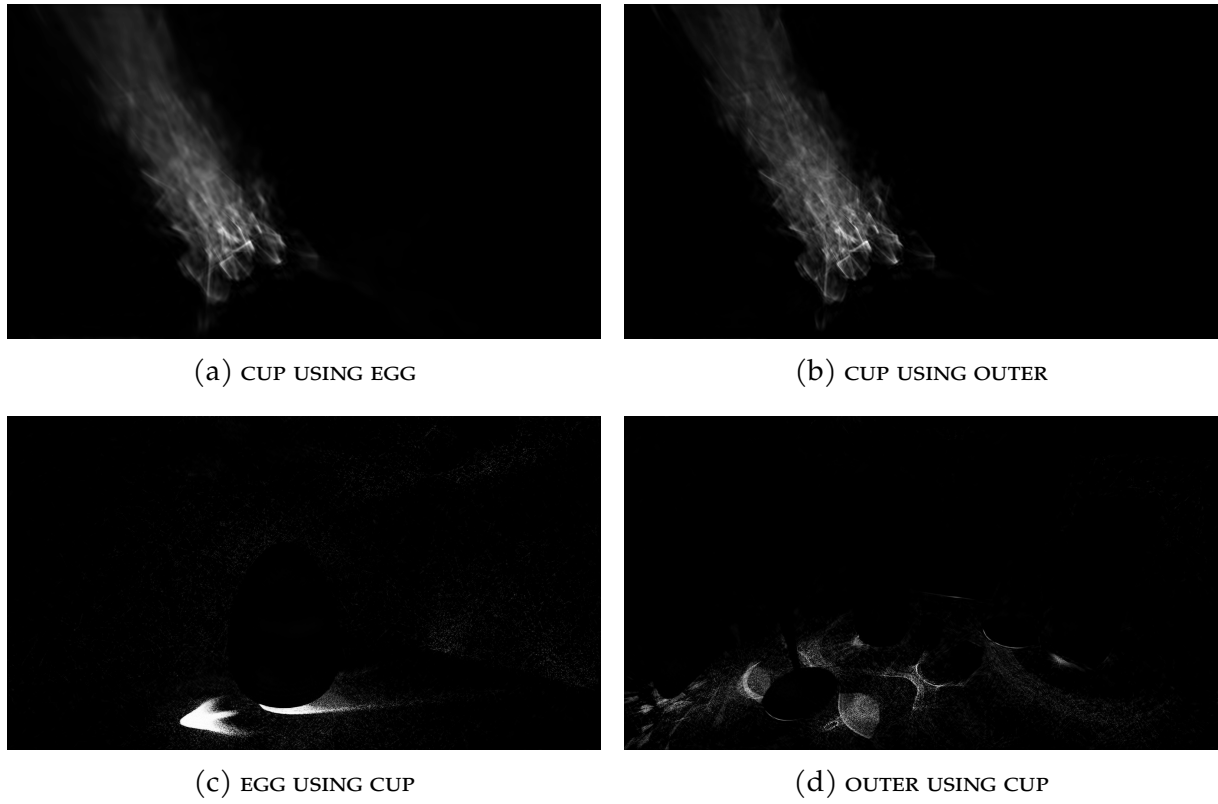


Figure 7.8 Denoising scenes with different coefficients.

	FLIP	VDP3	MSE	PSNR	SSIM
C-E	3.24e-02	6.35	3.15e-04	34.69	0.90
C-O	2.53e-02	7.15	2.43e-04	35.82	0.93
E-C	6.03e-02	5.08	1.69e-03	28.43	0.62
O-C	3.54e-02	6.86	7.63e-04	31.09	0.83

Table 7.14 Metric scores for the denoised scenes with different coefficients. A-B means scene A denoised with coefficients derived from scene B.

evaluated using FLIP. This improvement is attributable to the proficiency of curvelets in capturing edges and directional features, whereas the ReLAX results appeared blurred and exhibited residual noise. It should be noted that ReLAX is a general-purpose denoiser and was not specifically designed for caustics. Furthermore, the caustics in the DICE scene were subject to significant noise. Whilst denoising is the primary function of such an algorithm, the initial noise level must be manageable, and ideally, a form of importance sampling should be performed a priori.

Beyond improving the input data, ReLAX (and ReBLUR) can be configured to utilise a greater number of historical frames. This would likely enhance the quality of the denoised image at the cost of increased storage and computational overhead; however, this approach is less effective in dynamic scenes where caustics are changing. Although our method demonstrates superior performance, it necessitates a search for optimal coefficients. This process can be viewed as a form of training, which substantiates the performance gain over ReLAX and ReBLUR.

In low-frequency scenes such as EGG and TORUS, the caustics appeared comparable. Nevertheless, when ReBLUR was applied, the caustics became more distinct, but the curvelet-based images retained sharper detail. Artefacts inherent to the curvelet transform manifested as curvilinear structures rather than salt-and-pepper noise and had the potential to permeate other areas of the scene. This behaviour is anticipated, as errors in the curvelet coefficients naturally manifest as needle-like structures upon inverse transformation.

When measured against AI-based denoising, traditional denoisers prove to be less effective. OptiX demonstrates superior denoising performance across all scenes, resulting in sharp caustic patterns and noise-free images. For instance, in the OUTER scene (Figure 7.10), the OptiX result (centre) produces a cleaner image compared to the curvelet-denoised version (right), which exhibits residual curve-like artefacts across the entire area. As an AI-based denoiser, OptiX was trained on a corpus of tens of thousands of images from diverse 3D scenes. It is likely that this extensive training enables it to effectively recognise and reconstruct noisy signals, even in the presence of complex caustics.

The efficacy of our search algorithm (Algorithm 4) was highly dependent on the chosen metric. The performance of the metrics employed to determine the denoising coefficients exhibited considerable variation. Overall, FLIP performed well except in CUPS where the apparent noise is considerable and REFLECT where the scene appears over-exposed and loses detail. Conversely, the HDRVDP3 metric yielded less satisfactory results due to the presence of clustered speckle noise, as illustrated in Figure 7.9. MSE generally performed well, except in CUPS and DICE, where the image appeared noisy. Unexpectedly, PSNR's performance was lacking in scenes such as TORUS, EGG, DICE, OUTER, REFLECT and CUPS, despite its close relationship with MSE. SSIM emerged as the poorest

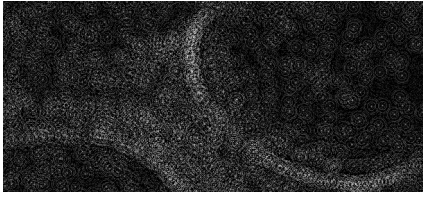


Figure 7.9 Clustered speckle noise in `DICE` with `VDP3`.

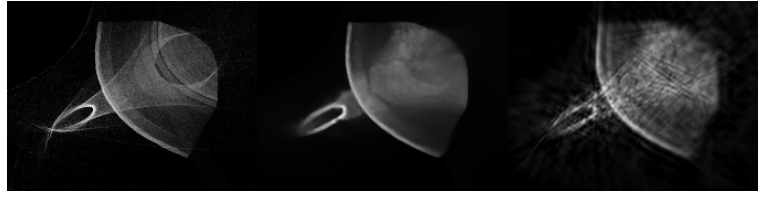


Figure 7.10 Left: Reference. Middle: OptiX. Right: Curvelet.

	<code>FLIP</code>	<code>VDP3</code>	<code>MSE</code>	<code>PSNR</code>	<code>SSIM</code>
Aggregate	1.12	1.02	1.93	1.24	1.47
FLIP	1.26	0.48	1.92	1.14	1.26

Table 7.15 Ratios for `DICE`. Higher (bold) is better.

performer. Searches employing `SSIM` yielded unsatisfactory outcomes in `CUPS` and `DICE` where the images are too noisy and no caustics can be easily identified, and `EGG`, where pixelation artefacts were visible. Additionally, in `OUTER` the image appeared blurred, and in `REFLECT` the image appeared underexposed, with caustic patterns being scarcely discernible.

Metric Reliability

Regarding the aggregate metrics, certain scenes exhibited higher scores across multiple metrics, yet the resulting image quality did not necessarily reflect this improvement. For instance, in Figure 7.5, the `FLIP` subfigure displays more defined caustics compared to the aggregate subfigure, despite the latter reporting higher scores across a greater number of metrics. Initially, we hypothesised that the discrepancy might stem from Algorithm 5, which can fail when improvements in individual metrics are positive but marginal, as opposed to fewer metrics exhibiting more substantial enhancements.

Upon closer inspection, a dry run of Algorithm 5 was conducted, and the resulting ratios and their differences were tabulated for the aggregate (`PSNR`) and curvelet-based metrics in Table 7.15. The data indicate that the `DICE` score (derived from the aggregate metric) underperforms specifically only in conjunction with the `FLIP` metric, and shows a greater improvement ratio for `VDP3`. Nevertheless, a qualitative assessment of the denoised images reveals that the output generated using the `FLIP` metric remains visually superior.

Search Start Location

Regarding the initial configuration during coefficient search, statistical analysis of the scenes we assessed indicates that commencing from the midpoint ensures the metric score attained is at least average or above. Although this may not yield the optimal

result, it provides a sound estimate for a starting point. It is crucial to understand that the true mean is anticipated to be within the confidence interval 95% of the time. Given that we evaluated eight intervals, there is a 0.95^8 (66.34%) probability that all intervals encompass their respective true mean.

Colour Images

Applying the existing coefficients to noisy coloured inputs yielded favourable results. As illustrated in Figure 7.6, the denoised images in RGB demonstrate comparable performance to the greyscale denoised images. Furthermore, there is a noticeable improvement across all metrics, reinforcing the notion that performing the search on the greyscale/luminance component effectively reduces the search time, allowing the coefficients to be seamlessly applied to each channel for colour denoising. The trade-off is the denoising time, which, according to the results, is approximately three times longer. However, a more robust outcome might be achieved by conducting the search for the thresholding coefficients on all channels individually and then denoising each channel separately, allowing for comparison with these results.

Image Translation

In terms of image translation using various transforms, the denoising performance remains largely equivalent. Moreover, the change in the metric score ratio remains practically constant across all increments, as illustrated in Figure 7.7. This indicates that the thresholding coefficients can be effectively reused across different views of a scene. At the very least, this appears to hold true for scenes denoised with the curvelet transform using the garrote thresholding function.

Reuse Across Scenes

In conclusion, it appears that the reuse of coefficients across multiple scenes is sometimes successful, though not consistently so. The strong similarity between CUP and OUTER (0.98) demonstrates a desirable outcome (Figure 7.8b), while the weak similarity between CUP and EGG (0.12) also yields positive results in the denoised image (Figure 7.8a), with CUP being only slightly blurred in comparison. It is unclear why WATER and EGG achieve a strong similarity score, given that both scenes feature different caustics content: WATER exhibits both low and high-frequency caustics, whereas EGG predominantly contains low-frequency caustics. The figures (Figures 7.8c and 7.8d) depicting EGG and OUTER denoised using CUP coefficients demonstrate some level of denoising performance.

Despite the uncertainty as to whether scene similarity directly correlates with similar coefficients, it is possible to reuse coefficients across different scenes. The results indicated that all coefficients provided a certain degree of denoising performance across all metrics, and the performance was unmatched when the denoising coefficients were applied to their intended scenes.

7.4 Summary

This chapter investigates the effectiveness of spectral transforms (Fourier, wavelet, and curvelet) in denoising caustics for Monte Carlo rendering, comparing them to the ReBLUR and ReLAX denoisers, and the AI-based OptiX denoiser. The study employs a systematic methodology involving search algorithms, thresholding functions, and image similarity metrics (FLIP, HDRVDP3, MSE, PSNR, SSIM) to optimise denoising coefficients. Results show that spectral methods, particularly curvelets, outperform ReLAX and ReBLUR in preserving high-frequency caustic details but generally fall short of OptiX's superior noise reduction. Experiments confirm that coefficients derived from greyscale images can effectively denoise colour images and remain robust under spatial transformations like translations and flips. Scene similarity analysis via cosine metrics suggests some coefficient reusability across scenes, though performance varies. While spectral denoising requires offline tuning, it offers promising detail preservation, making it a viable alternative in scenarios where AI-based methods may oversmooth critical features. The study also highlights the need for less noisy inputs to preserve specific caustic characteristics, as aggressive denoising can blur fine details or leak curve-like structures even when spectral transforms are employed.

8 Importance

Chapter 7 examined spectral denoising and compared its performance with state-of-the-art denoising techniques, including ReBLUR, ReLAX, and OptiX. The analysis demonstrated that spectral denoisers outperform ReBLUR and ReLAX, though the AI-based OptiX generally achieves superior results. However, OptiX remains ineffective in denoising caustic features at 16 spp for certain scenes, while increasing the sample rate beyond 16 spp would significantly compromise real-time rendering performance.

This chapter focuses on enhancing and extending the CLT introduced in Chapter 5, ensuring it remains unbiased while reducing noise at the same 16-spp sample rate. Modifications to CLT incorporate importance sampling techniques, including Caustics-Optimised Light Tracing (COLT), which resembles NEE in path tracing by explicitly sampling specular materials, and Importance Maps which are textures that store per-textel probability values generated at each light source. These maps are populated through multiple methods, accounting for factors such as camera position, distance from light sources, and occlusion.

The evaluation employs the same test scenes and metrics from Chapter 7, with the addition of the LPIPS metric. The aggregate metric, previously a linear combination of all measures, is revised to include LPIPS and to highlight discrete improvements without considering the ratio of enhancement. Using this updated metric, the new CLT variants are assessed, with comparative plots of noise levels at varying sample rates and 16-spp images for qualitative analysis, generated using Forge.

Chapter 7 noted occasional discrepancies between image similarity metrics and visual assessments. This chapter confirms that such inconsistencies are not exclusive to denoising and investigates the issue by generating modified noisy and reference images, including blurred versions, to determine whether noise reduction adversely affects metric reliability.

Finally, the study attempts to denoise CLT output using the new importance-based methods, partially leveraging the framework from Chapter 7. The denoising performance is evaluated by comparing results with the original non-importance-based approach.

8.1 Caustics-Optimised Light Tracing

The primary objective in this chapter is to improve CandelaDXR's Light Tracer so that it produces less noisy samples at 16 spp. The first technique we are looking at is COLT. COLT consists of using importance sampling to prefer paths that sample dielectric materials that allow for reflection and refraction. From the light source or at each diffuse

point, a parameter α is used to determine whether or not to sample a specular material with probability α . If a specular material is to be sampled, a random object with a specular material is selected. Finally, a random point on the object is generated. Then, using the area-based version of the rendering equation, a ray is spawned towards this point. If a specular object is not sampled, a ray is spawned randomly using a cosine-distributed PDF (to match Lambert's law). However, in this case, if a specular object is encountered, ray termination occurs to prevent double counting. This process is repeated at every intersection, and may terminate using russian roulette. This method is similar to the NEE method used to sample lights in path tracing, except that no extra rays are generated (as in shadow rays).

This technique is effective when used in scenes that have a small number of specular objects with respect to diffuse objects. If a scene is composed mostly of specular objects, the probability of hitting specular objects is high and this algorithm starts becoming equivalent to the naive light tracer.

8.2 Importance Map Light Tracer

Another variation with respect to the light tracer revolves around the use of importance maps. Importance maps in this context are 2D textures modelling a probability distribution. Each texel is associated with a value that represents the probability of it being selected. The importance map light tracer uses this map to control the direction of a spawned ray from a light source. This variation involves multiple passes.

First, the importance map is populated with some method (discussed in 8.2.1). Then, a compute shader pass is employed to perform Gaussian blur. This pass is optional and its necessity depends on how the importance map was populated. The CDF is then computed in the subsequent three passes. The first one performs a prefix sum, an operation that for each texel finds the sum of all the previous texel values. For simplicity assume a 1D array of 5 values, $[1, 2, 3, 4, 5]$. The prefix sum in this case is $[1, 3, 6, 10, 15]$. To turn this into a valid CDF, the prefix sum array must be normalised. This is done by dividing all the elements in the array with the last element. In this example, all elements need to be divided by 15 resulting in $[1/15, 3/15, 6/15, 10/15, 15/15]$. A final pass is performed that simply writes 1 to the last element, this is performed only for numerical stability in this light tracing shader as occasionally the result is not exactly 1 due to numerical errors.

8.2.1 Population Methods

Three methods have been used to populate the importance map. Prior to further modifications to the importance map, a base map representing a cosine distribution designed

to match Lambertian surfaces is used for importance maps representing area lights. The reason is that when sampling an area light, when an importance map is not used, cosine distribution is normally applied. Therefore, the directions sampled outside of the importance map will match those of the importance map by default. Equation 8.4 is the PDF of the cosine distribution parameterised using cartesian coordinates and is used to approximate the cosine distribution within the importance map. C refers to the CDF, A to surface area onto the imaginary importance map, r is the distance from the origin of the light source to the the world coordinates p of the texel of the importance map, z is the z component of p and $\hat{n}$ is the normal at p . θ is the angle between the normal at the light source and the direction to p , and θ_p is the angle between the normal at the p and the direction to the light source. To compute the exact probability for each texel, Equation 5.14 could be used, however that is a heavy computation. Additionally, precision is not particularly important at this stage. The approximation suffices and the CDF is later calculated discretely using the prefix sum and normalisation passes. The methods below, which are used to manipulate the importance map, multiply their resulting map with the base map.

$$C = \frac{1}{\pi} \int \cos \theta d\omega \quad (8.1)$$

$$\frac{dC}{dA} = \frac{1}{\pi r^2} \cos \theta \cos \theta_p \quad (8.2)$$

$$\cos \theta = \cos \theta_p = \frac{p \cdot \hat{n}}{r} = \frac{z}{r} \quad (8.3)$$

$$\frac{dC}{dA} = \frac{z^2}{\pi r^4} \quad (8.4)$$

Distance-based Importance

The first method, which we refer to as `LT_IMP_DIST`, weighs the map using the squared distances from the light source to the intersecting mesh as seen by the light source, for each texel. The distance is calculated by first using a rasterisation pass to compute the G-buffer, and then a compute shader to compute the distances. This importance map gives far away objects a higher probability of being sampled than closer ones. Additionally, empty areas are marked with the lowest probability (zero for single point sources).

Distance- and Camera-based Importance

The second method, which we refer to as `LT_IMP_DIST_CAM`, uses Equation 8.5. It weighs the importance map using distance squared similar to the first method however it is also proportional to the angle between the camera and the intersection point, and inversely proportional to the distance squared between the camera and the intersection point.

Therefore, it is more likely for light to fall on areas of the mesh to which the camera is looking directly at and to which the camera is closer to.

$$P_{weight} = \frac{\hat{n} \cdot (CameraPos - IntersectionPos) \|IntersectionPos - LightPos\|^2}{\|CameraPos - IntersectionPos\|^3} \quad (8.5)$$

Distance- and Camera-based Importance with Culling

The third method, which we refer to as `LT_IMP_DIST_CAM_RT`, uses the exact same function as the second method but further employs ray tracing to create a texture representing a mask. The mask filters the importance map to be limited to unoccluded areas between the intersection point and the camera. This ensures that light samples are minimised in areas where the light reaches a mesh, but the camera cannot see this mesh.

8.2.2 CDF Sampling

Depending on the light source type, the sampling strategy varies, but the general process is similar. The goal is to find a direction from the light source that improves the distribution of photons in a light tracer. A light source is selected at random, a point on it is chosen, and an initial direction is sampled, for instance, using a cosine distribution for area lights. We then determine if this direction intersects the virtual CDF plane. If it does not, light tracing continues normally. If it does intersect, we must account for the probability of this hit in the Monte Carlo estimator (this probability is obtained by dividing P_r from Equation 5.14 by π). Subsequently, a new direction is sampled according to the PDF stored in the CDF plane, as detailed in Algorithm 6. This algorithm inverts the CDF via a binary search to find a texel, providing a sample and its corresponding PDF (see Table 8.1 for variable definitions). Finally, Algorithm 7 is used to generate a ray from the light source in the direction of the sampled texel, updating the ray's direction and calculating the necessary coefficients for rendering. Figure 8.1 illustrates a 2D projection of the importance map overlaying the hemisphere from the light source's view, demonstrating why a base map is constructed to account for the shrinking area off-centre in the hemisphere and to apply the cosine distribution.

8.2.3 Interpolation of PDFs

The linear interpolation property of probability density functions (PDFs) enables useful applications in rendering. Given two valid 2D PDFs, a and b , with dimensions $m \times n$ where $\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} b_{ij} = 1$ and each element is a positive number,

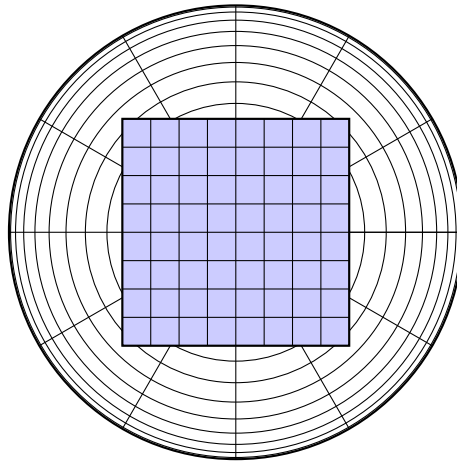


Figure 8.1 CDF plane inside the hemisphere subtended by a light source

Table 8.1 Variable Definitions for Algorithms 6 and 7

Variable	Definition
CDF	CDF array
cdfIndex	Index to select a specific CDF map
coeff	Geometric term: $\cos \theta / \text{distance}^2$
$\cos \theta$	Cosine between light normal and ray
distance	Distance from ray origin to world point
impPdf	PDF from importance map sampling
meta	Importance map metadata
meta.plane	Virtual plane parameters
meta.probSelectingRect	Importance map selection probability
pdf	PDF value
pt	2D intersection point on CDF plane
r	Uniform random number in $[0,1)$
ray	Ray with origin and direction
seed	Random number generator seed
texel	Sampled 2D texture coordinates
texelArea	Area of a single texel
texelIndex	1D texel index from CDF search
unitLightNormal	Light surface normal vector
uv	2D texture coordinates of the texel
worldPt	3D world point for sampled texel
w, u, v	Orthonormal basis from light normal

Algorithm 6 Importance Map Sampling

```

1: function SAMPLEIMPORTANCEMAP(seed, cdfIndex)
2:   CDF  $\leftarrow$  cdf[cdfIndex]
3:                                      $\triangleright$  Check if CDF contains valid light sources
4:   if CDF is invalid (no visible lights) then
5:     pdf  $\leftarrow$  0.0
6:     return ((0,0), 0)
7:   end if
8:                                      $\triangleright$  Generate random number in [0, 1)
9:   r  $\leftarrow$  rand_next(seed)
10:                                      $\triangleright$  Find pixel index using inverse CDF sampling
11:   texelIndex  $\leftarrow$  binarySearchCDF(r, cdfIndex)
12:   if texelIndex is invalid then
13:     return ((0,0), 0)
14:   end if
15:                                      $\triangleright$  Convert to 2D coordinates and compute PDF
16:   uv  $\leftarrow$  indexTo2D(texelIndex)
17:   pdf  $\leftarrow$  CDF[texelIndex] - CDF[texelIndex - 1]
18:   return (uv, pdf)
19: end function

```

their weighted combination $c_{ij} = a_{ij}w + b_{ij}(1 - w)$ for $0 \leq w \leq 1$ also forms a valid PDF. This is proven by Equation 8.10, which shows that $\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} c_{ij} = 1$.

$$\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} b_{ij} = 1 \quad (8.6)$$

$$\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} c_{ij} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (a_{ij}w + b_{ij}(1 - w)) \quad (8.7)$$

$$w \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (a_{ij}) + (1 - w) \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (b_{ij}) \quad (8.8)$$

$$w(1) + (1 - w)(1) = 1 \quad (8.9)$$

$$\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} c_{ij} = 1 \quad (8.10)$$

Figure 8.2 illustrates this property with three 4×4 PDFs, where c represents the interpolation of a and b with weight $w = 0.7$. The colour scheme visually encodes the interpolation: red tones indicate stronger influence from a , blue tones from b , and their mixture produces pink hues in c . Additionally, colour intensity corresponds to probability values, with darker shades representing higher probabilities.

This interpolation property extends naturally to CDFs and is particularly valuable for area light sampling. Consider a triangular area light where instead of using a

Algorithm 7 Construct ray using CDF if CDF virtual plane is hit

```

1: function CONSTRUCTRAYUSINGCDF(seed, ray, pdf, coeff, unitLightNormal,
   cdfIndex)
2:   meta  $\leftarrow$  impMeta[cdfIndex]  $\triangleright$  Get metadata of importance map
3:   (w, u, v)  $\leftarrow$  constructOrthonormalBasis(unitLightNormal)
4:    $\triangleright$  Check if ray intersects light plane
5:   if not intersectsLightPlane(ray, w, meta.plane) then
6:     return false
7:   end if
8:   pt  $\leftarrow$  projectToLightPlane(ray, w, u, v, meta.plane)
9:   if not isWithinLightPlaneBounds(pt, meta.plane) then
10:    return false
11:  end if
12:  (texel, impPdf)  $\leftarrow$  sampleImportanceMap(seed, cdfIndex)
13:  if impPdf = 0.0 then
14:    return false
15:  end if
16:  worldPt  $\leftarrow$  texelToWorldSpace(texel, w, u, v, meta.plane)
17:   $\triangleright$  Update ray direction and compute coefficients
18:  ray.Direction  $\leftarrow$  normalized(worldPt - ray.Origin)
19:  distance  $\leftarrow$  length(worldPt - ray.Origin)
20:  cos $\theta$   $\leftarrow$  dot(w, ray.Direction)
21:  coeff  $\leftarrow$   $\frac{\cos\theta}{\text{distance}^2}$ 
22:   $\triangleright$  Final PDF: importance PDF  $\times$  selection probability / texel area
23:  texelArea  $\leftarrow$  computeTexelArea(meta.plane)
24:  pdf  $\leftarrow$  impPdf  $\times$   $\frac{\text{meta.probSelectingRect}}{\text{texelArea}}$ 
25:  return true
26: end function

```

single CDF based on the triangle’s midpoint, we precompute CDFs at each vertex. During Monte Carlo sampling, when selecting a point on the triangle, we can interpolate the vertex CDFs using barycentric coordinates, similar to normal interpolation, to obtain a spatially varying CDF. This approach yields more accurate importance sampling in light tracing, as the sampling distribution adapts to the triangle’s geometry rather than assuming uniform characteristics across the surface.

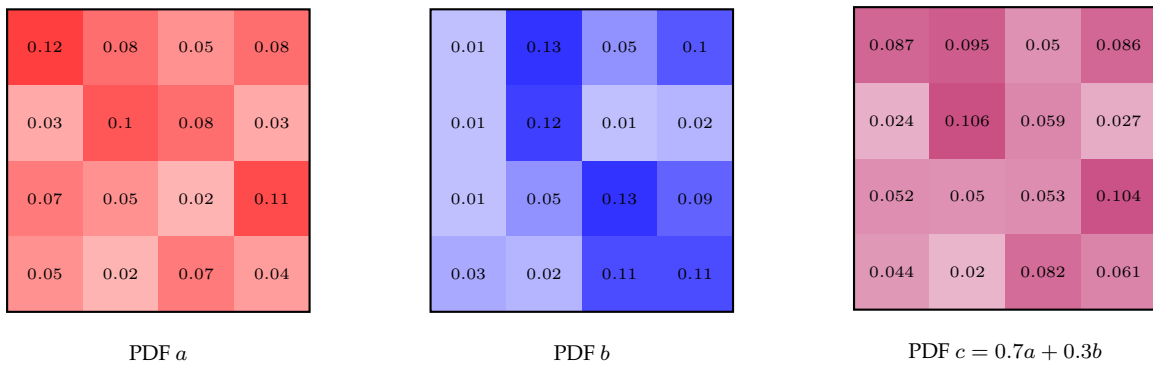


Figure 8.2 PDF interpolation example with $w = 0.7$

8.3 Limitations

Importance maps do not perform well with scenes consisting of many lights. Storing a map for every light in a scene would require a large amount of memory and processing power. Importance maps need to be recomputed whenever the position or orientation of a light source changes, or whenever the camera changes. One possible way to account for many lights would be to randomly select a single light source per draw call. This contrasts with the alternative of having each shader thread randomly choose a light, which forces the entire set of lights, and importance map for each light, to be accessible to the shader for a single draw call. This mitigates all the issues discussed above, however a single frame on its own would only contain the contribution from a single light source. This solution would yield an unbiased result given an infinite number of frames. However this is not useful for real-time rendering where we can only afford to generate a small number of frames. This approach only works well in offline rendering.

8.4 Methodology

This study introduces and evaluates a range of light-tracing-based methods, as outlined in the previous sections. The primary objective is to assess the efficacy of these methods in reducing variance when rendering caustics. To achieve this, image similarity metrics are employed to compare rendered images against a reference image. Metrics such as

FLIP, SSIM, MSE, PSNR, HDRVDP3, and LPIPS are utilised, providing a comprehensive evaluation framework.

A diverse set of scenes featuring caustics is essential for this evaluation, as caustics vary in nature; some exhibit sharp, high-frequency characteristics with curvature, while others are more uniform and low-frequency. The selected scenes, which include those used in the Spectral Denoising chapter, are a mixture of both types. This ensures a robust evaluation across a range of caustic behaviours.

To ensure reproducibility, the stochastic nature of Monte Carlo renderers is accounted for by recording the seed for each rendered image. This allows for consistent replication of the experiment under the same conditions.

The evaluation is conducted through a systematic procedure, involving the generation of multiple image buffers, the creation of reference images, and the comparison of rendered images against these references using the specified metrics. An aggregate metric is also derived to summarise the results, providing a holistic view of the performance of each shader across all metrics.

As a final step, a subset of scenes computed with the importance-based methods are chosen and denoised using the methods in Chapter 7. A quantitative and qualitative analysis is performed in order to understand whether the performance of the denoisers, especially the spectral-based denoisers and OptiX increases, especially whether they are able to retain important curve-like structures (an important characteristic feature in caustics) details better.

8.5 Experimental Setup

This section presents the specific configuration and parameters used to implement the methodology outlined in Section 8.4. It details the test scenes and shaders employed, the procedure for generating and comparing images, the criteria for denoising evaluation, and the analytical framework for interpreting the results. The following subsections provide a comprehensive overview of the experimental configuration.

8.5.1 Scenes

The scenes used in this experiment include those from the Spectral Denoising chapter (Chapter 7) and from the Light Tracing chapter (Section 5.3), ensuring consistency with prior work. These are shown in Table 8.2, and the shaders used to render these scenes are reiterated in Table 8.3. An additional scene, `EGG_CLOSEUP`, presented in Figure 8.4a, was introduced to provide a detailed view of the caustics present in `EGG`. This selection ensures a mix of high-frequency (sharp, curved) and low-frequency (uniform)

caustics, allowing for a comprehensive evaluation of the methods under different conditions. The reference shader column in the table shows which shader has been used to generate it. The lights used in the scenes are also noted for more understanding with regards to how the importance maps have been constructed.

Scene	Reference Shader	Lights
CUP	LT_IMP_DIST_CAM	Spot
CUPS	LT_IMP_DIST_CAM	Directional
DICE	LT_IMP_DIST_CAM	Directional
REFLECT	LT_IMP_DIST_CAM	Directional
OUTER	LT_IMP_DIST_CAM	Directional
TORUS	LT_IMP_DIST_CAM	Directional
EGG	LT_OPT	Area
EGG_CLOSEUP	LT_OPT	Area
WATER	LT	Spot

Table 8.2 Scene setup configuration

Shader	Description
PT	Path Tracer
LT	Light Tracer (Algorithm 3)
LT_OPT	Light Tracer with importance for specular materials (COLT - Section 8.1)
LT_IMP_DIST	Light Tracer using distance-based importance maps (Section 8.2.1)
LT_IMP_DIST_CAM	Light Tracer using distance and camera-based importance maps (Section 8.2.1)
LT_IMP_DIST_CAM_RT	Light Tracer using distance and camera-based importance maps with ray traced culling (Section 8.2.1)

Table 8.3 Shader descriptions

8.5.2 Image Generation

For each rendered image, four buffers are output: diffuse, specular, caustics, and radiance. The primary focus is on the caustics buffer, with the radiance buffer considered supplementary.

A reference image is generated for each scene using 4 million spp, produced with a specific shader detailed in the accompanying table. For every combination of scene, buffer, and shader, nine images are produced. Each image in the sequence has a sample count ranging from 4^0 to 4^8 spp; where the number of samples per pixel quadruples for each subsequent image as shown in Table 8.4. This is a design choice intended to cut the variance in half. These images share a common seed, while the reference image uses a distinct seed to ensure independence.

Image Number (n)	Samples per Pixel ($s = 4^n$)
1	$4^0 = 1$
2	$4^1 = 4$
3	$4^2 = 16$
4	$4^3 = 64$
5	$4^4 = 256$
6	$4^5 = 1024$
7	$4^6 = 4096$
8	$4^7 = 16384$
9	$4^8 = 65536$
12	$4^{11} = 4194304$

Table 8.4 Samples per Pixel for Each Image Number

8.5.3 Image Comparison and Metrics

Each rendered image is loaded and evaluated in three states: raw, tone-mapped with ACES, and tone-mapped with ACES plus gamma correction. The primary focus is on the tone-mapped images without gamma correction. Each transformed image is compared to the reference using the specified metrics (FLIP, SSIM, MSE, PSNR, HDRVDP3, and LPIPS), and the results are recorded.

An aggregate metric is derived to summarise the findings. This metric counts the number of improvements across all metrics for each shader. Given six metrics and five shaders (excluding the reference), 30 comparisons are performed, resulting in an aggregate score ranging from 0 to 30. A score of 30 indicates universal improvement across all metrics, while a score of 0 signifies no improvement or worse performance. The aggregate metric weighs all metrics equally, though the weighting can be adjusted to prioritise specific metrics if necessary.

8.5.4 Visualisation and Analysis

The results are visualised through a series of plots, with a graph generated for each combination of scene, buffer type, and metric. With nine scenes, seven metrics (including the aggregate), and four buffer types, a total of 252 graphs are produced. Due to the volume of data, the discussion primarily focuses on the aggregate metric, with individual metrics examined as needed for clarification or deeper insight.

8.5.5 Denoising

Four scenes are selected for denoising evaluation: CUPS, DICE, REFLECT, and OUTER. The analysis computes metric scores for both the noisy 16 spp caustic buffer input and the denoised outputs from ReBLUR, ReLAX (in both accumulating and non-accumulating

variants), as well as the spectral denoisers. These scores are evaluated against the five metrics and the aggregate metric established in Chapter 7.

The results are compared quantitatively with both the noisy inputs and the denoised outputs from Chapter 7, which did not incorporate importance-based sampling methods. For qualitative assessment, the denoised buffers are visually inspected and discussed in detail.

8.5.6 Verdict and Discussion

The best-performing shader for each scene is identified based on the aggregate metric. Consistency in performance across multiple scenes strengthens the case for a shader's superiority. However, particular emphasis is placed on improvements achieved with lower sample-per-pixel (spp) counts, as real-time rendering constraints often limit spp to 16 or fewer. This ensures practical relevance for real-time applications.

The discussion in Section 8.7 provides an interpretation of the results, exploring observed relationships and offering explanations for the findings. This analysis contextualises the performance of the shaders and their implications for rendering caustics effectively.

8.6 Evaluation

Figures 8.5 and 8.4a present the reference images for each scene. These images were rendered at 4 million spp using a hybrid rendering approach that combines path tracing and light tracing, as specified in Table 8.2. Path tracing was employed to connect specular objects with the pinhole camera, as such paths cannot be generated by a light tracer. The corresponding reference caustic buffers, which rely exclusively on light tracing, are shown in Figures 8.6 and 8.4b. The cropped radiance buffers are displayed in Figures 8.7 and 8.4c, while the blurred radiance buffers are presented in Figures 8.8 and 8.4d.

Figures 8.9 through 8.17 show the caustic buffers for all scenes at 16 spp, as generated by each shader. The aggregate metrics for the caustic buffers are illustrated in Figures 8.18 and 8.4e. Furthermore, the aggregate metrics for comparisons performed after blurring both the noisy and reference images are shown in Figures 8.19 and 8.4f, and those for comparisons after downsampling all images by 25% are given in Figures 8.20 and 8.4g. The aggregate metrics for cropped versions of the original images, focusing on areas containing caustics, are presented in Figures 8.21 and 8.4h. Finally, Table 8.5 details the computation time in milliseconds for each shader per frame, where the total time is the sum of the `PT` time and that of any `LT`-based shader.

In all scenes except EGG, EGG_CLOSEUP, and WATER, the score for LT_IMP_DIST_CAM increased steadily with more iterations, surpassing all other shaders upon reaching 4096 spp across all metrics. This indicates that it is the shader which demonstrates consistent improvement across the majority of metrics. It eventually achieves the maximum aggregate score of 30 in these scenes, signifying improvement across every metric. In the EGG, EGG_CLOSEUP, and WATER scenes, the highest scorer is LT beyond 4096 iterations.

The first few iterations are most important for realtime rendering. Considering the first 16 spp; For CUP, LT_IMP_DIST_CAM_RT scored highest during the first 4 iterations surpassed by LT_IMP_DIST_CAM thereafter. In CUPS, LT_OPT scored highest while in DICE, it is surpassed by PT starting from 16 spp. In REFLECT, LT_OPT, LT and LT_IMP_DIST_CAM_RT scored higher. In OUTER and TORUS, PT scored higher. Finally, in EGG, EGG_CLOSEUP, and WATER, LT and LT_OPT scored higher.

There are unexpected results. The path tracer is only capable of generating caustics in EGG and EGG_CLOSEUP because it is only able to connect rays from specular objects to area light sources. This limitation is evident in Figure 8.11, where directional lighting in DICE prevents the formation of caustics. However, in DICE, all metrics determine that the path tracer (PT) performed better at 16 spp. This result is consistent (to a lesser degree) whether the image is gamma corrected or not tone mapped at all. When the result is gamma corrected, at 16 spp, only LPIPS determines path tracing to be worse than most other shaders. With the raw image however, only MSE determines path tracing to be the worse result. This problem appears also in CUPS at 64 spp (frame 3), in OUTER during the first three frames and in TORUS during the first four frames.

Additionally, MSE shows an abnormal spike at this sample point. This is outlined in Figure 8.3.

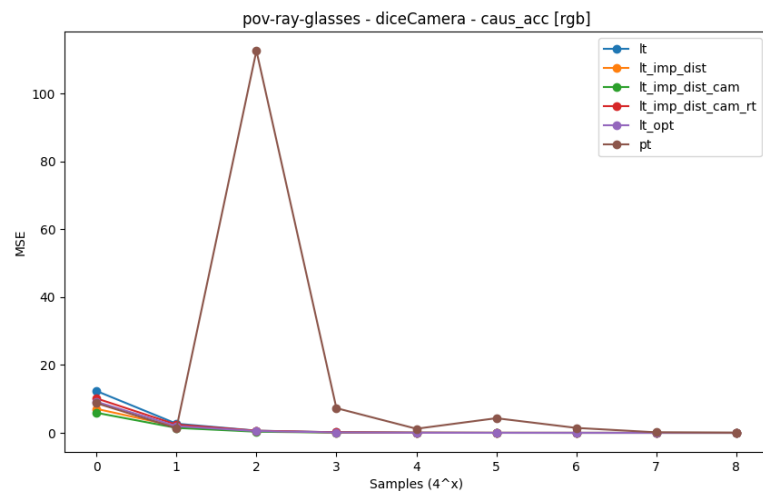


Figure 8.3 Spike in MSE with PT with raw render

8.6.1 Variations

The blurred versions are illustrated in Figures 8.19 and 8.4f. These demonstrate that `LT_IMP_DIST_CAM` outperforms other shaders at 16 spp in the `CUP`, `CUPS`, `DICE`, `REFLECT`, and `TORUS` scenes. In `OUTER`, it outperforms other shaders from 64 spp onwards. In `EGG` and `EGG_CLOSEUP`, `LT_OPT` and `LT` perform better, while in `WATER`, `LT` outperforms at all sample counts. In `DICE`, the path tracer scores higher only up to 4 spp and unlike in Figure 8.18, its performance declines with more samples per pixel.

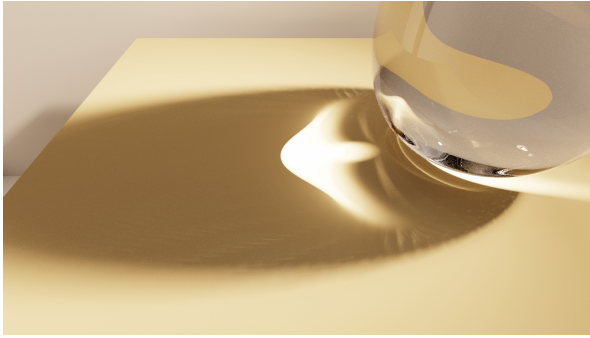
Up to 16 spp, `PT` dominates only at 1 spp in `CUPS`, and only up to 4 spp in `DICE`. In `REFLECT`, `LT_IMP_DIST_CAM_RT` performs better up to 4 spp, after which `LT_IMP_DIST_CAM` scores better at 16 spp. In `OUTER`, `LT_OPT` is superior, whereas in `TORUS`, `LT_IMP_DIST_CAM` scores better starting from 4 spp. In `EGG`, `LT_OPT` scores higher, and in `WATER`, `LT` performs best.

The downsampled versions, shown in Figures 8.20 and 8.4g, are broadly similar to the blurred versions, with slight discrepancies in `OUTER` and `TORUS`. In `OUTER`, `LT_OPT` dominates up to 64 spp, rather than 16 spp. In `TORUS`, `LT` performs better at 256 and 1024 spp, instead of only at 256 spp.

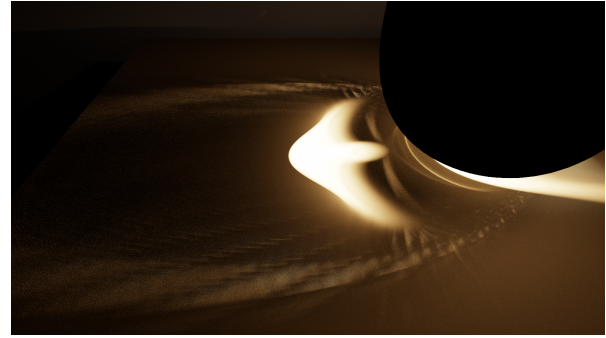
Figures 8.21 and 8.4h present the cropped versions. `LT_IMP_DIST_CAM` dominates at higher spp in all scenes except `EGG` and `WATER`, where `LT_OPT` and `LT`, respectively, perform better. At lower sample counts, up to 16 spp, the best-performing shaders are varied, with trends that share similarities to the original results. In `CUP`, `CUPS`, and `OUTER`, `LT_IMP_DIST_CAM` performs best at 16 spp; in `DICE` and `TORUS`, `PT` is superior. In `EGG` and `WATER`, `LT_OPT` and `LT` perform best respectively.

scene	pt	lt	lt_opt	lt_imp_dist	lt_imp_dist_cam	lt_imp_dist_cam_rt
cup	1.02	2.11	3.42	2.56	2.79	2.42
cups	1.65	2.06	3.10	2.88	4.18	3.73
reflect	1.36	2.06	3.07	2.88	4.08	3.62
dice	1.52	2.10	3.10	2.95	3.49	3.21
outer	1.88	2.65	3.65	3.86	4.14	3.95
torus	1.21	1.80	2.54	2.14	2.37	2.15
egg	1.10	8.38	5.00	5.36	5.50	5.50
water	1.12	5.02	4.30	6.10	5.50	4.82
egg_closeup	1.38	6.20	4.15	4.84	4.78	4.86

Table 8.5 Shader performance per frame (approximate in ms) per scene. Hardware: Intel i7-8700K, NVIDIA GeForce RTX 2080 Ti



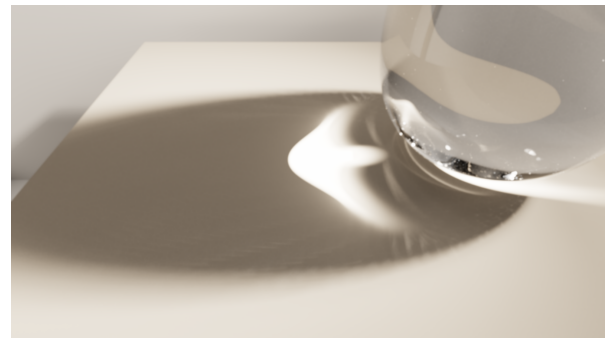
(a) Radiance



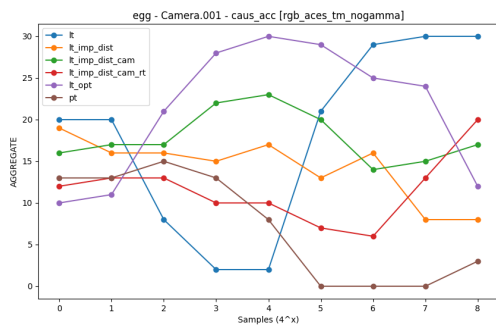
(b) Caustics



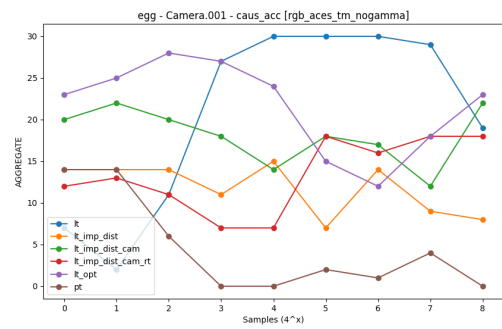
(c) Cropped (Rotated)



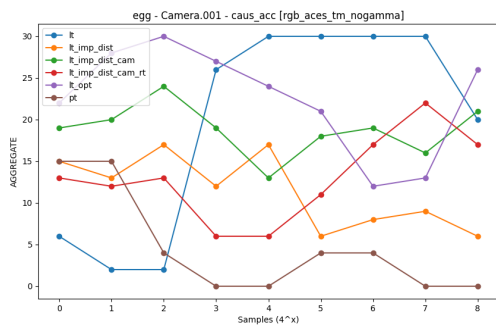
(d) Blurred



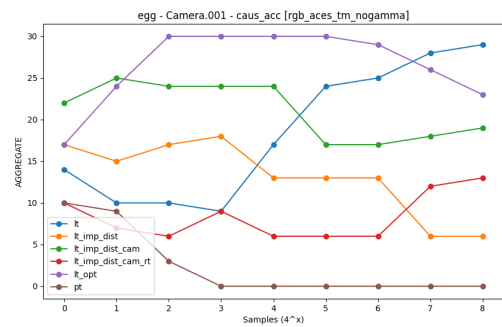
(e) Aggregate



(f) Aggregate - Blurred

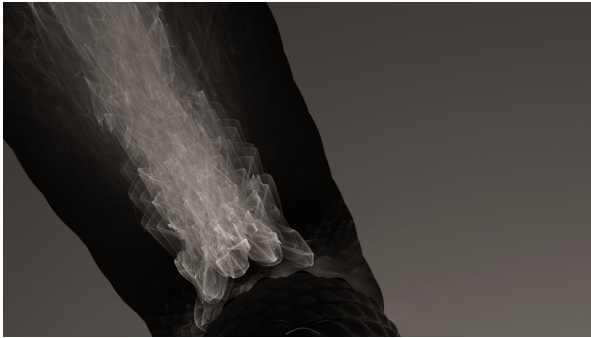


(g) Aggregate - Downsampled 25%

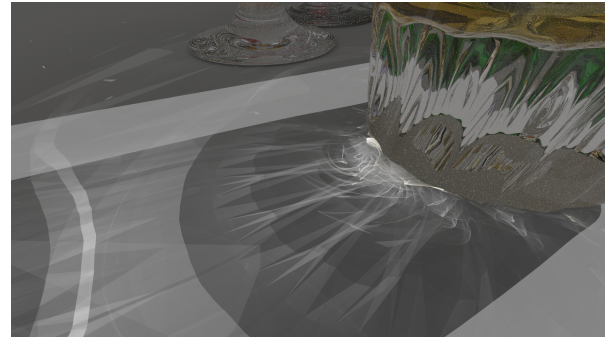


(h) Aggregate - Cropped

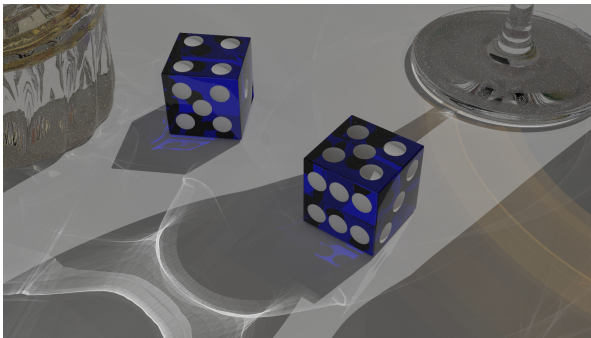
Figure 8.4 EGG_CLOSEUP buffers, variations and plots



(a) CUP



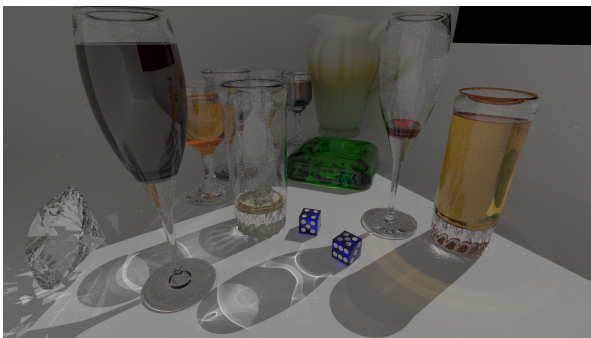
(b) CUPS



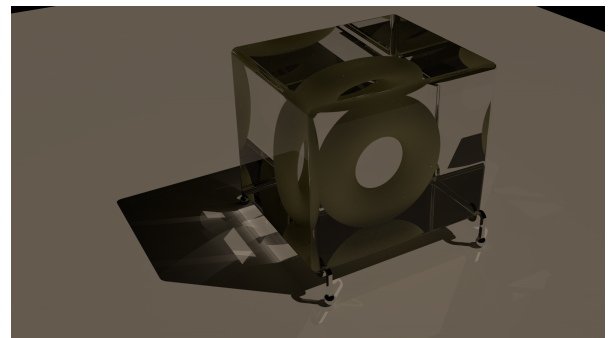
(c) DICE



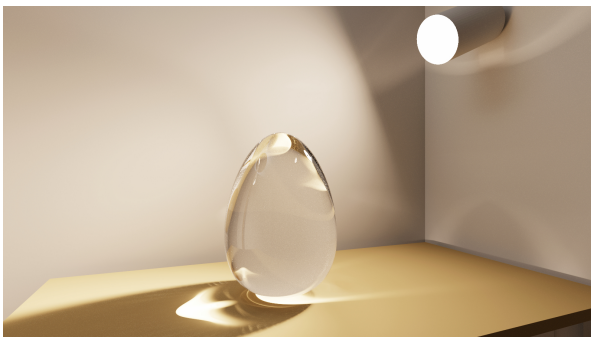
(d) REFLECT



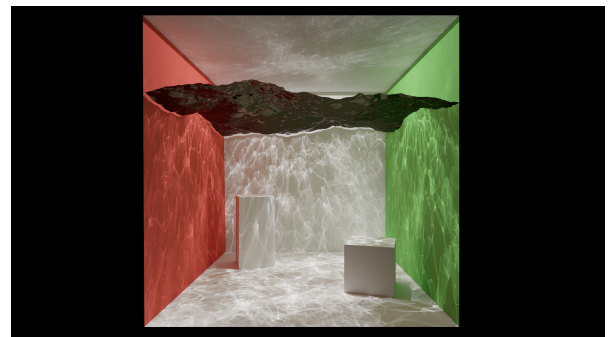
(e) OUTER



(f) TORUS

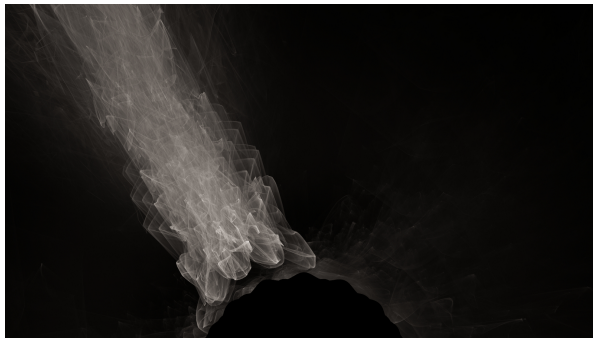


(g) EGG

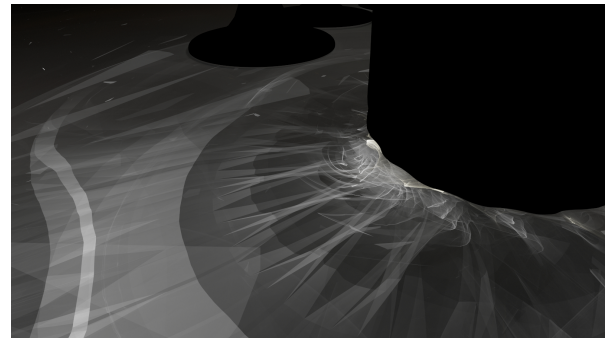


(h) WATER

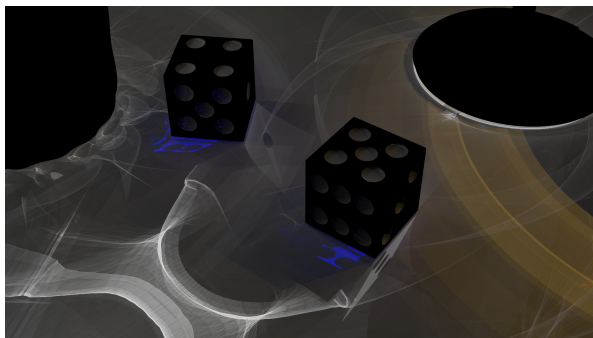
Figure 8.5 Radiance Buffers



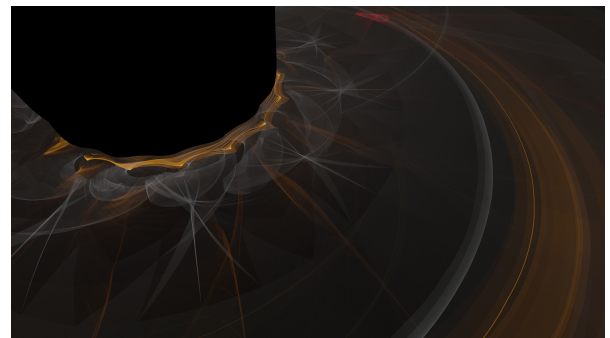
(a) CUP



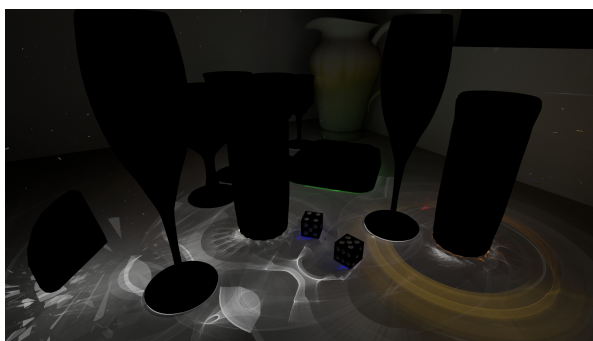
(b) CUPS



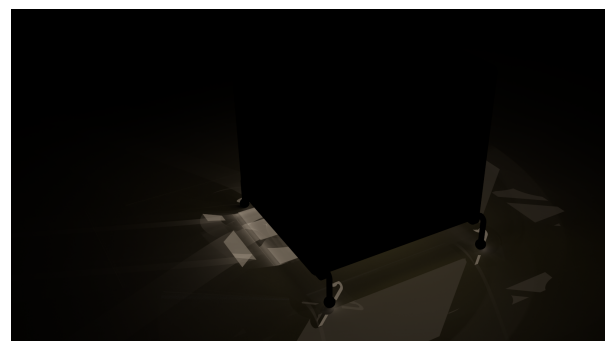
(c) DICE



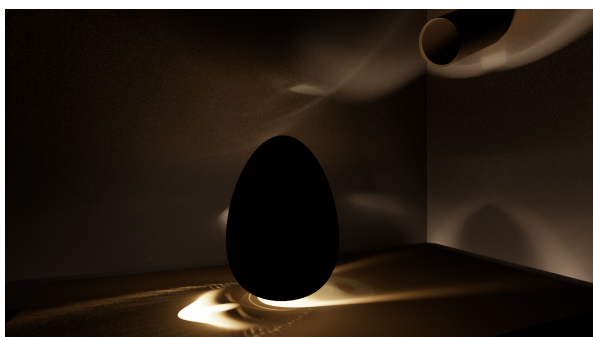
(d) REFLECT



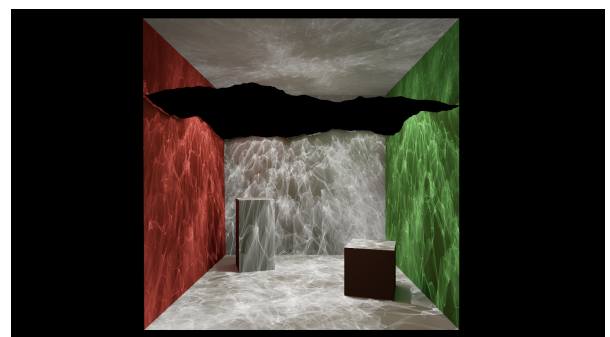
(e) OUTER



(f) TORUS

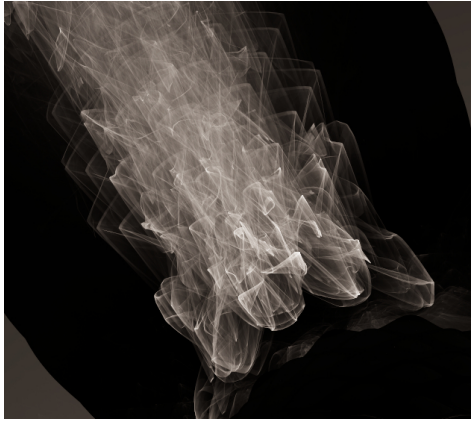


(g) EGG

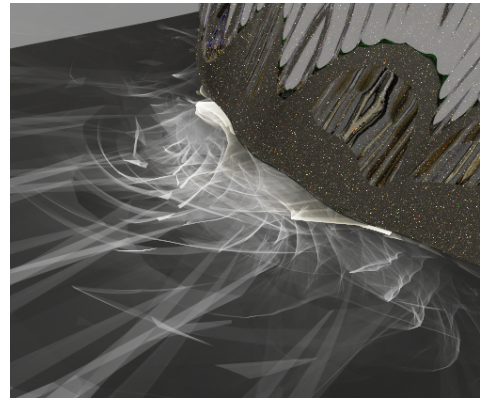


(h) WATER

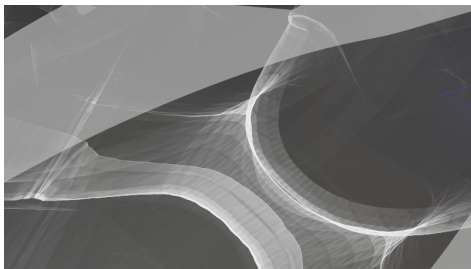
Figure 8.6 Caustics Buffers



(a) CUP



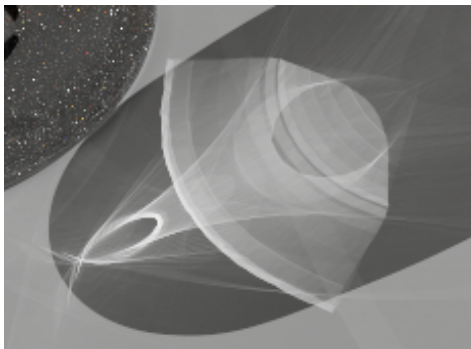
(b) CUPS



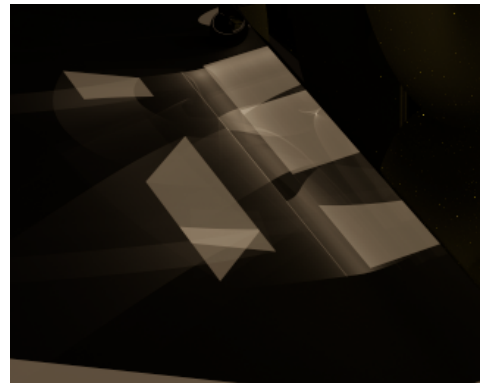
(c) DICE



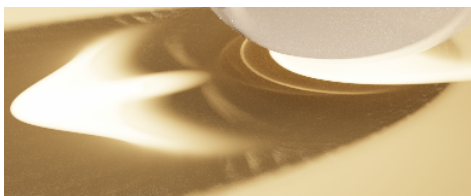
(d) REFLECT



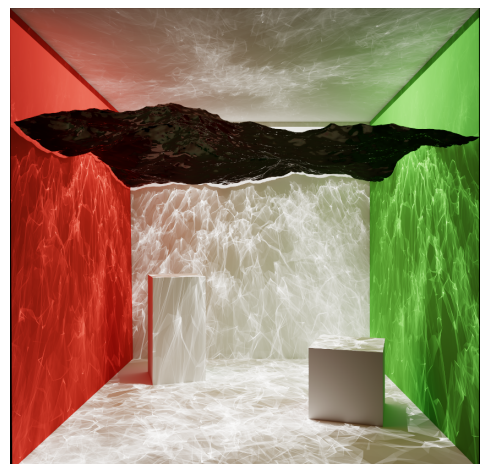
(e) OUTER



(f) TORUS



(g) EGG

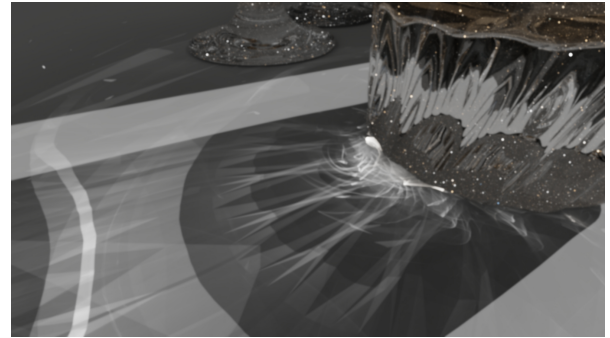


(h) WATER

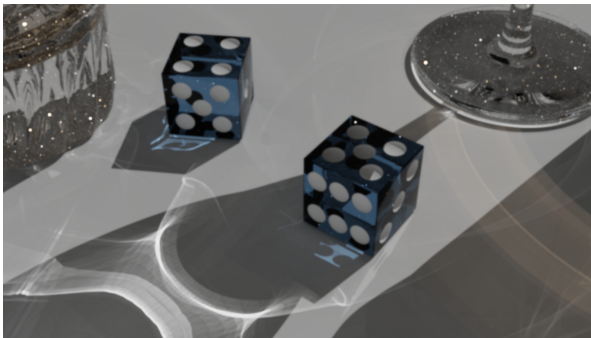
Figure 8.7 Cropped Radiance Buffers



(a) CUP



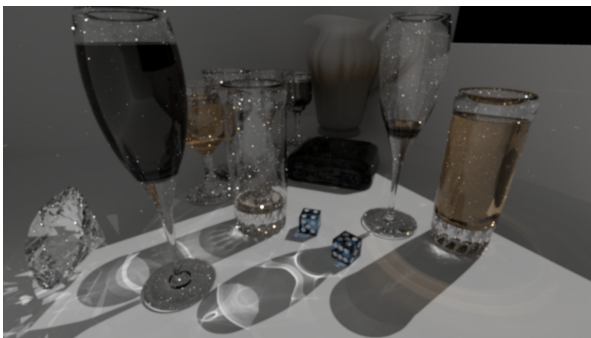
(b) CUPS



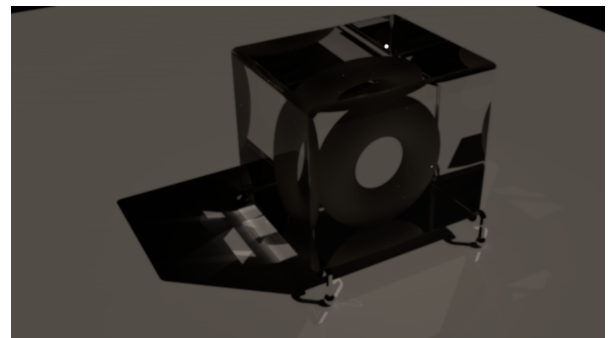
(c) DICE



(d) REFLECT



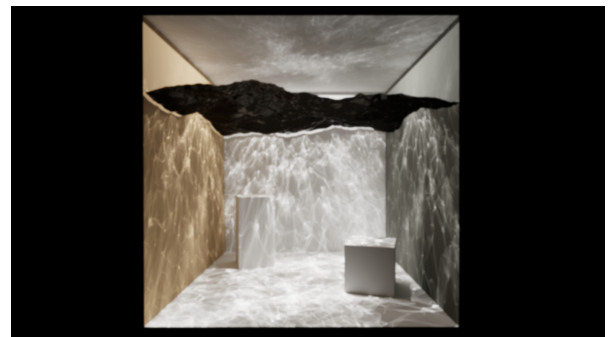
(e) OUTER



(f) TORUS

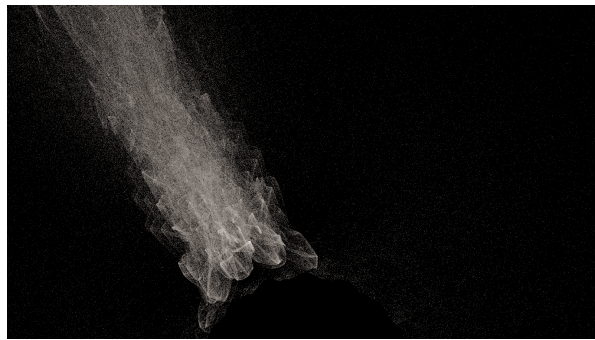


(g) EGG

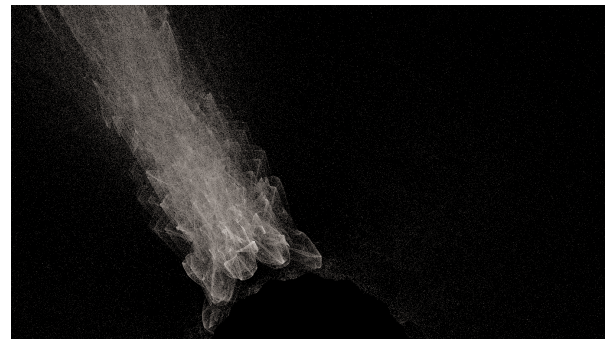


(h) WATER

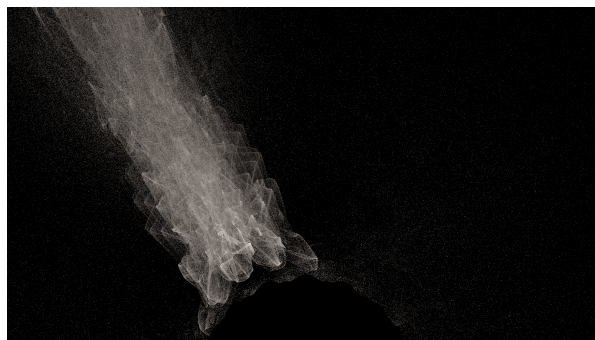
Figure 8.8 Blurred Radiance Buffers



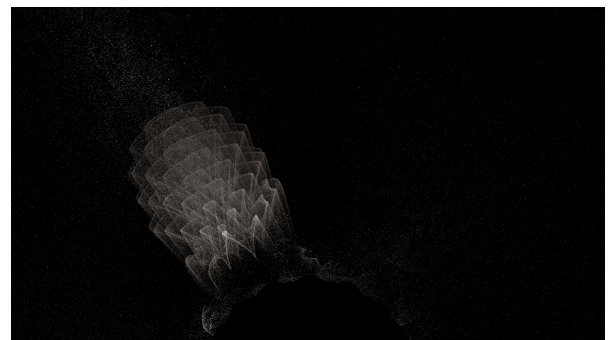
(a) LT



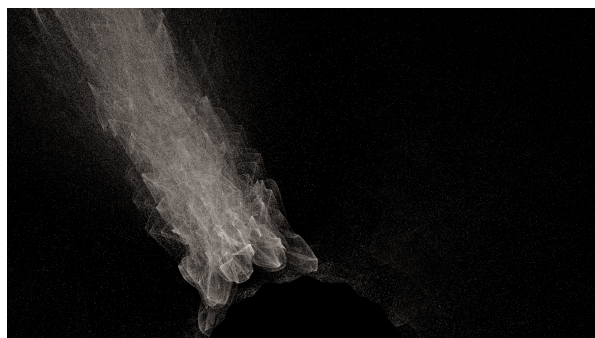
(b) LT_IMP_DIST



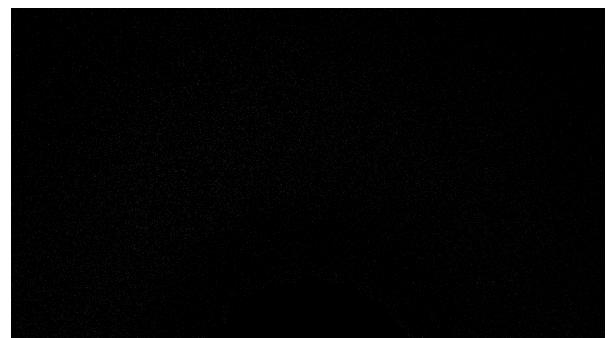
(c) LT_IMP_DIST_CAM



(d) LT_IMP_DIST_CAM_RT

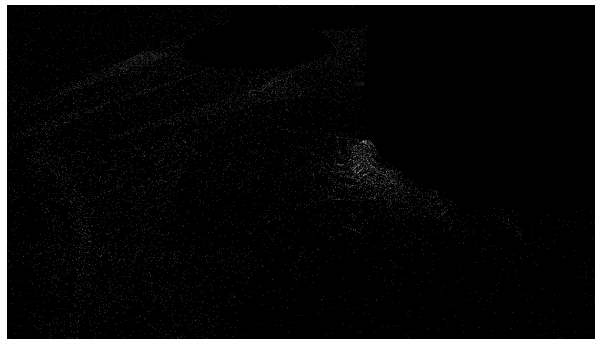


(e) LT_OPT

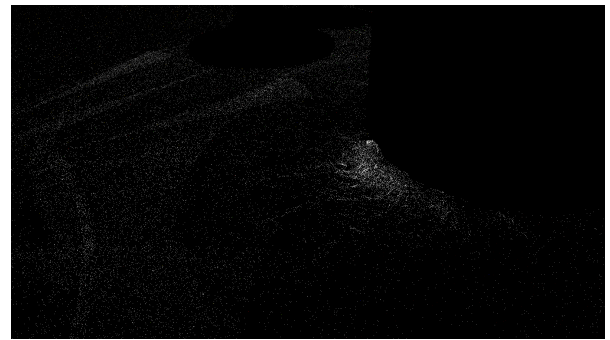


(f) PT

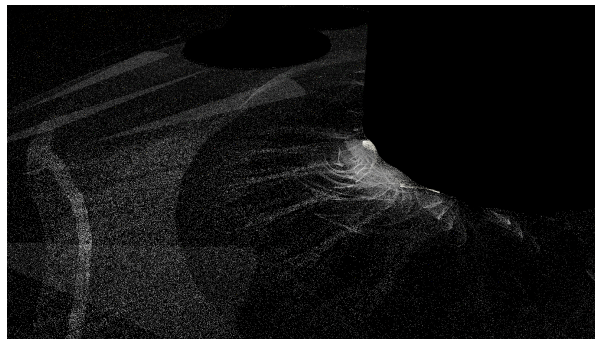
Figure 8.9 CUP renders



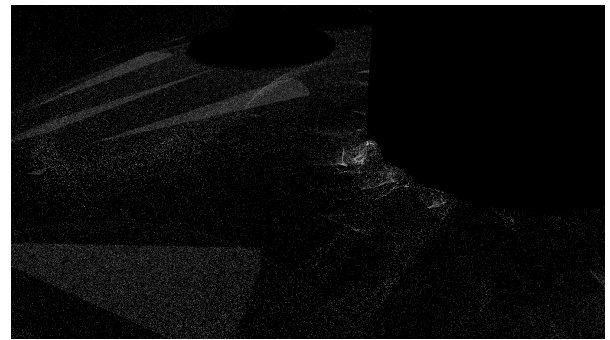
(a) LT



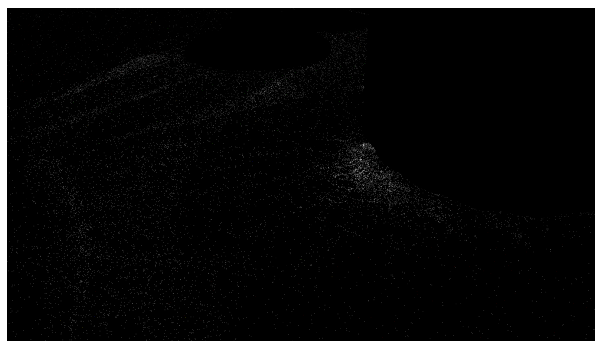
(b) LT_IMP_DIST



(c) LT_IMP_DIST_CAM



(d) LT_IMP_DIST_CAM_RT

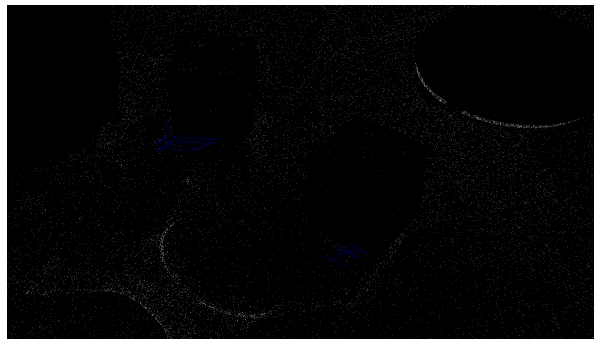


(e) LT_OPT

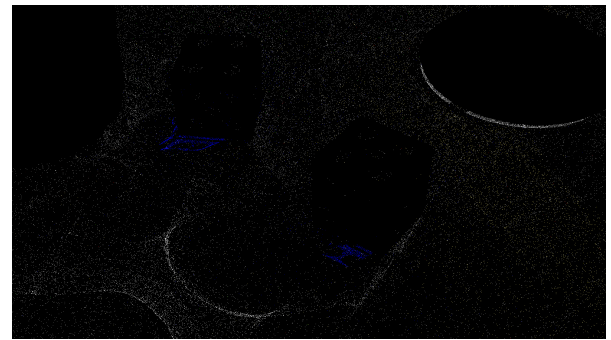


(f) PT

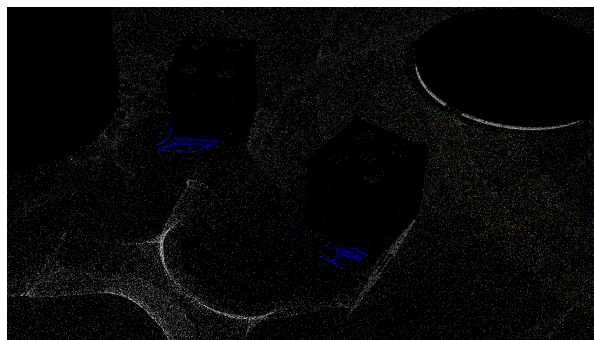
Figure 8.10 cups renders



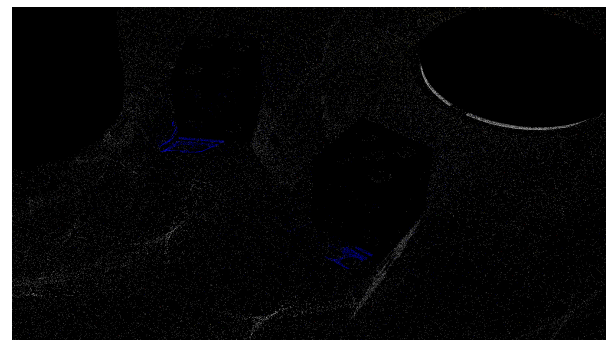
(a) LT



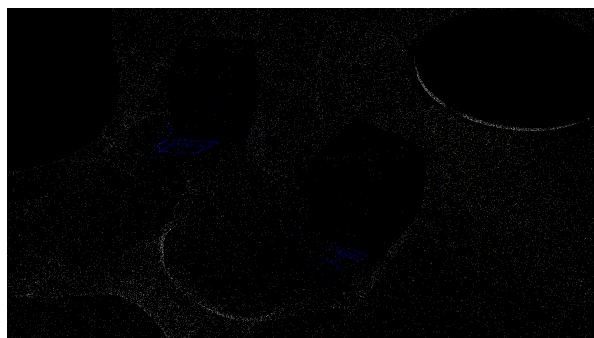
(b) LT_IMP_DIST



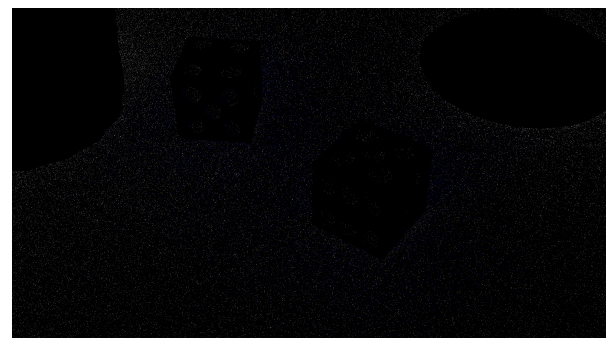
(c) LT_IMP_DIST_CAM



(d) LT_IMP_DIST_CAM_RT

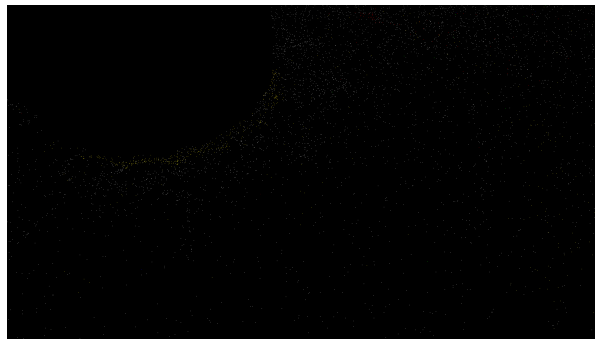


(e) LT_OPT

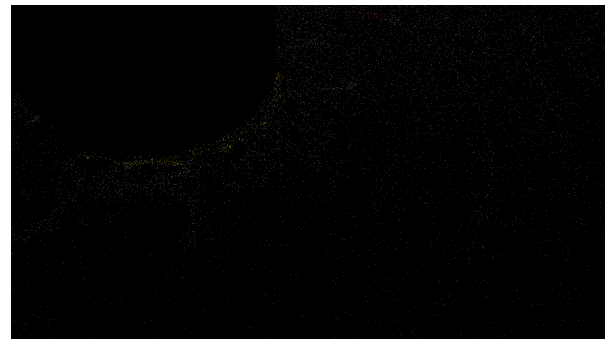


(f) PT

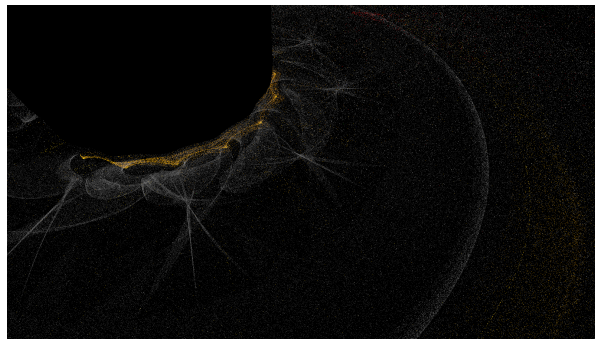
Figure 8.11 DICE renders



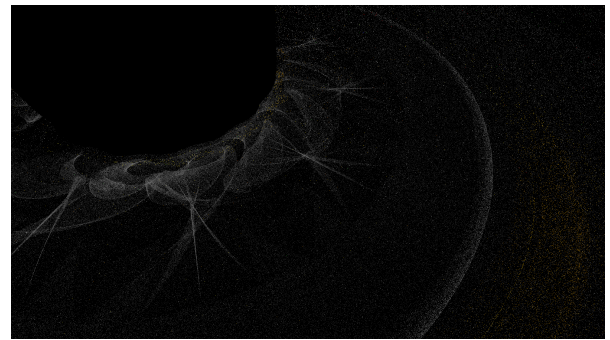
(a) LT



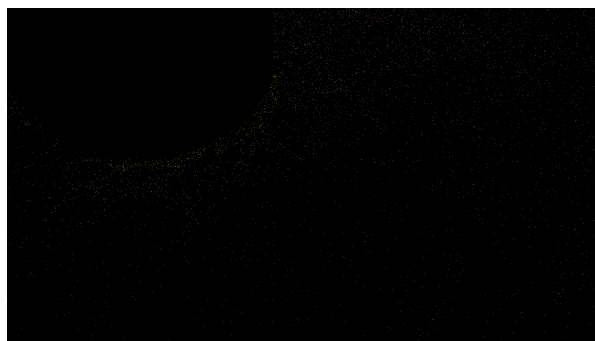
(b) LT_IMP_DIST



(c) LT_IMP_DIST_CAM



(d) LT_IMP_DIST_CAM_RT

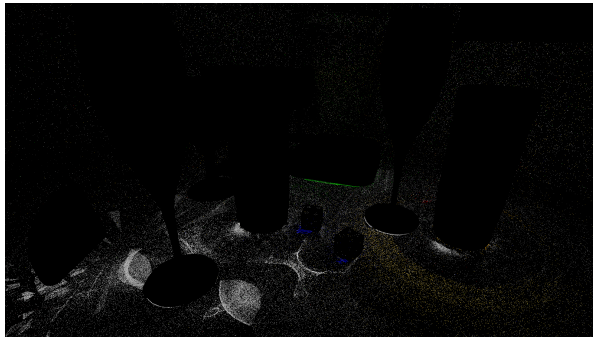


(e) LT_OPT

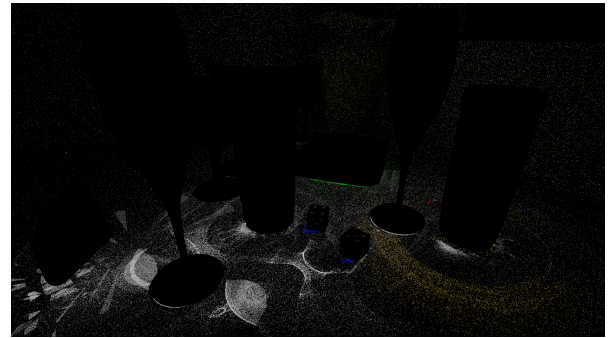


(f) PT

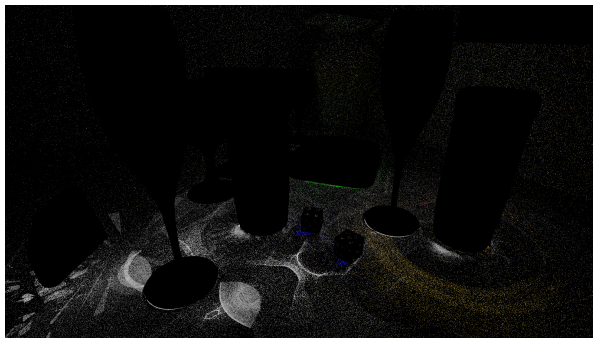
Figure 8.12 REFLECT renders



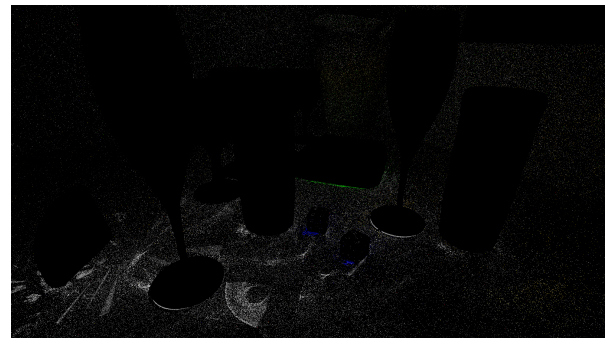
(a) LT



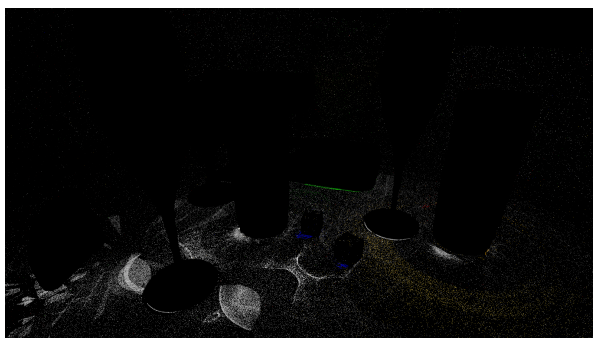
(b) LT_IMP_DIST



(c) LT_IMP_DIST_CAM



(d) LT_IMP_DIST_CAM_RT

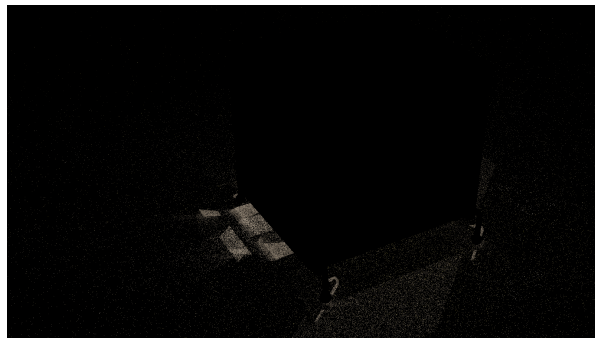


(e) LT_OPT

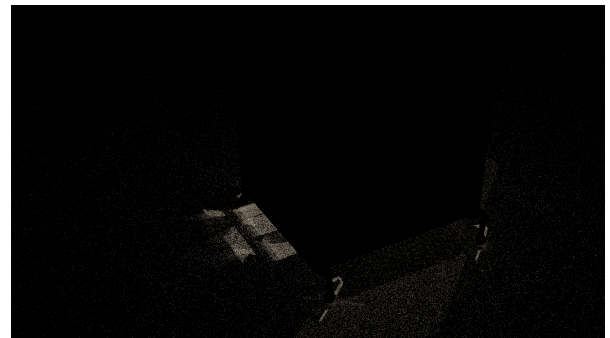


(f) PT

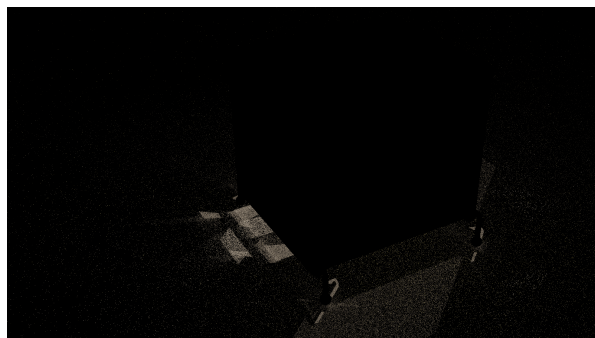
Figure 8.13 OUTER renders



(a) LT



(b) LT_IMP_DIST



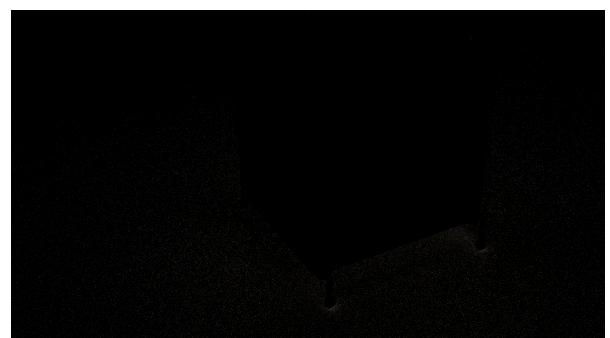
(c) LT_IMP_DIST_CAM



(d) LT_IMP_DIST_CAM_RT

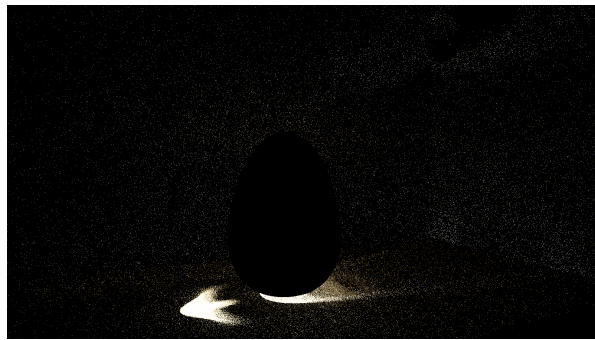


(e) LT_OPT

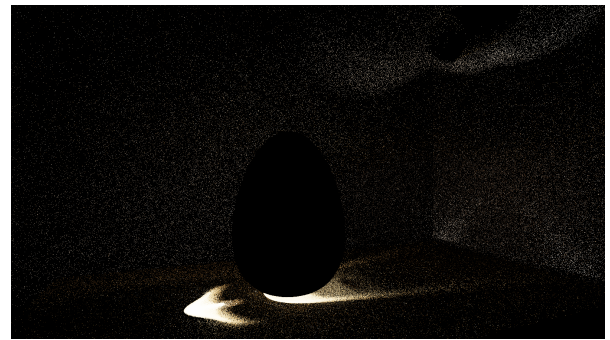


(f) PT

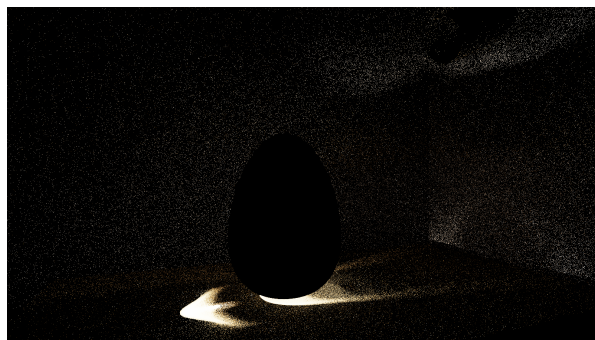
Figure 8.14 TORUS renders



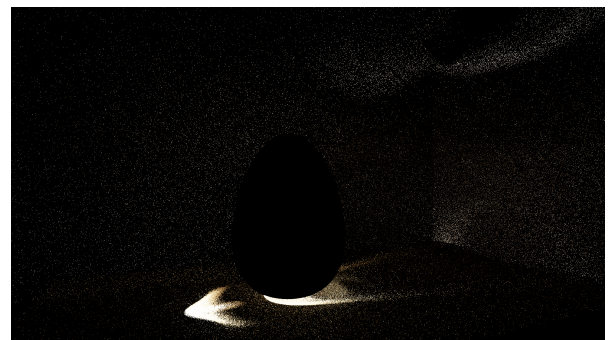
(a) LT



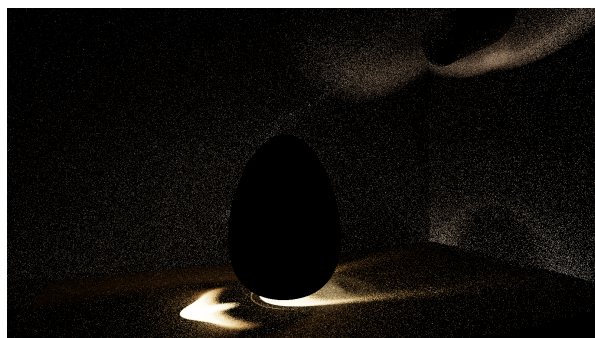
(b) LT_IMP_DIST



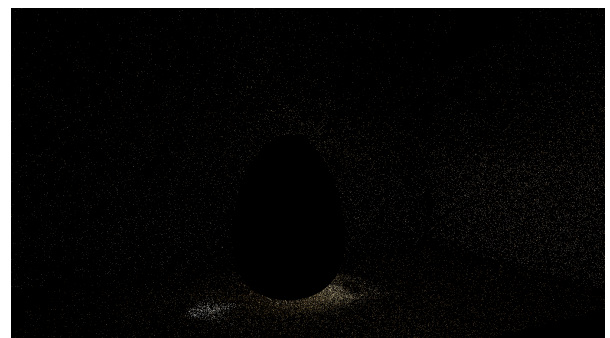
(c) LT_IMP_DIST_CAM



(d) LT_IMP_DIST_CAM_RT



(e) LT_OPT



(f) PT

Figure 8.15 EGG renders

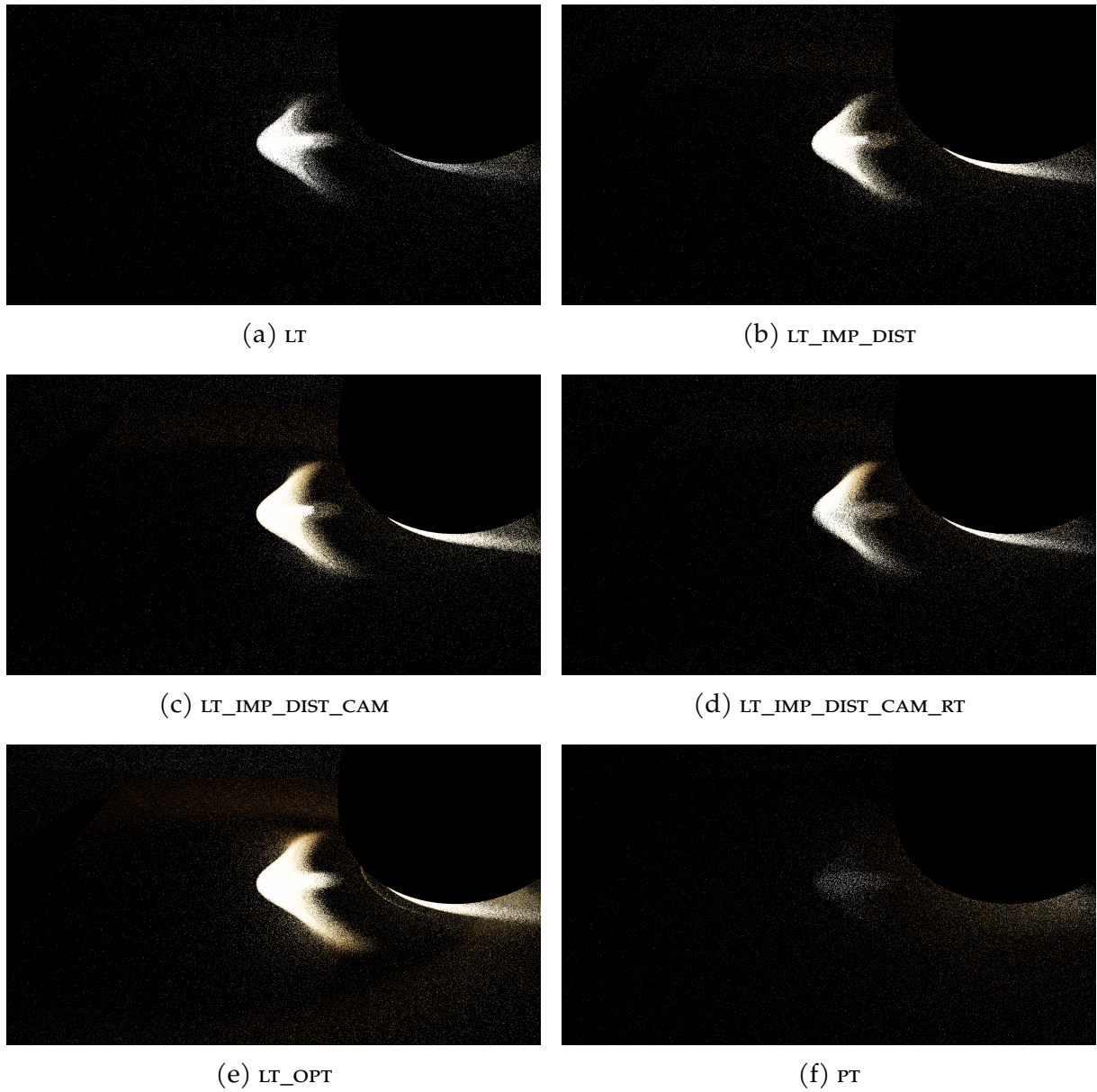
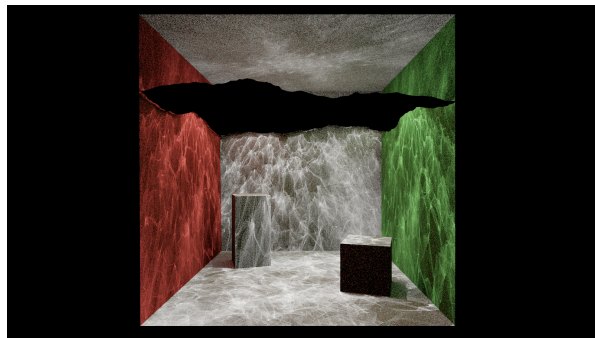
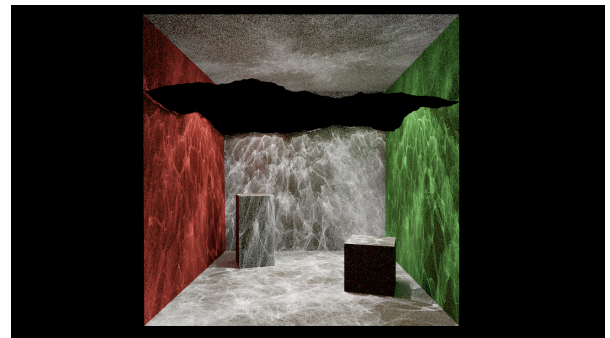


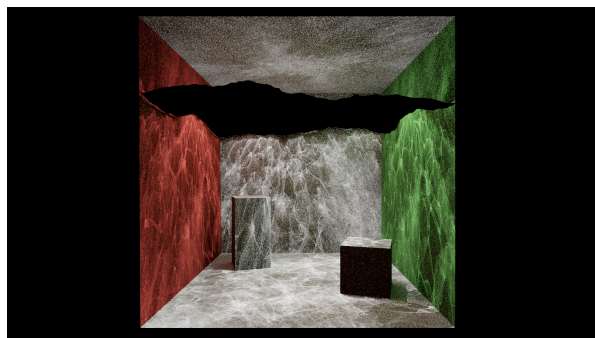
Figure 8.16 EGG_CLOSEUP renders



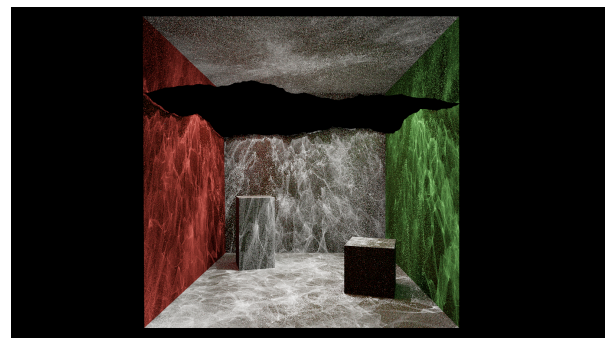
(a) LT



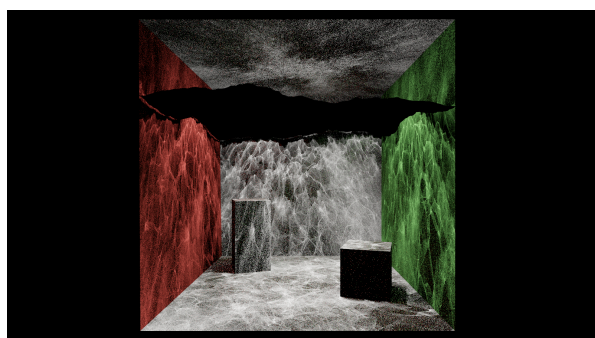
(b) LT_IMP_DIST



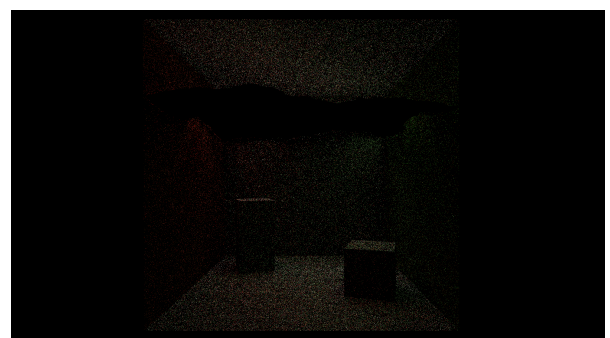
(c) LT_IMP_DIST_CAM



(d) LT_IMP_DIST_CAM_RT

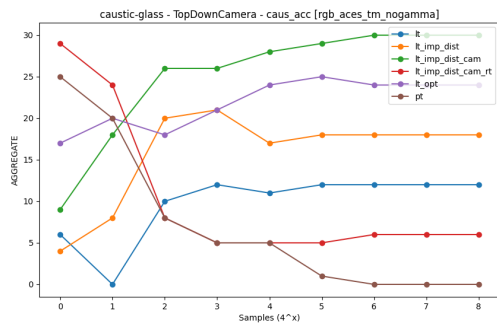


(e) LT_OPT

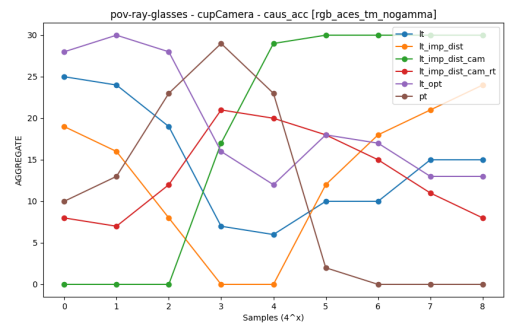


(f) PT

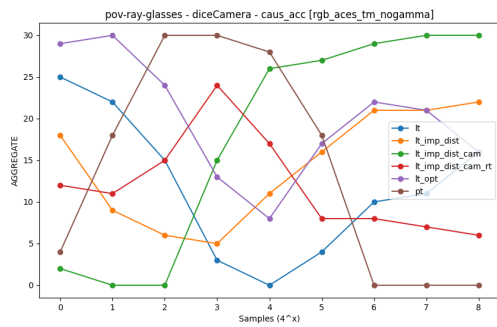
Figure 8.17 WATER renders



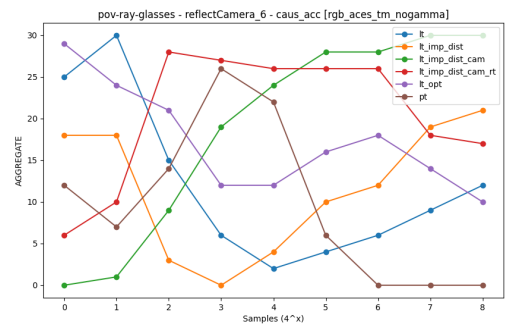
(a) CUP



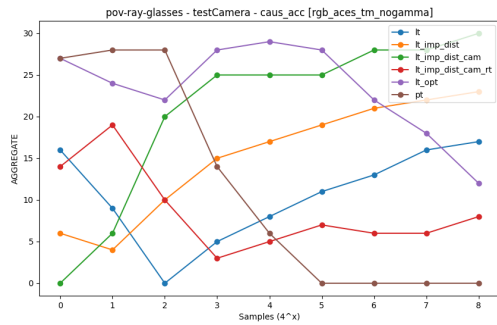
(b) CUPS



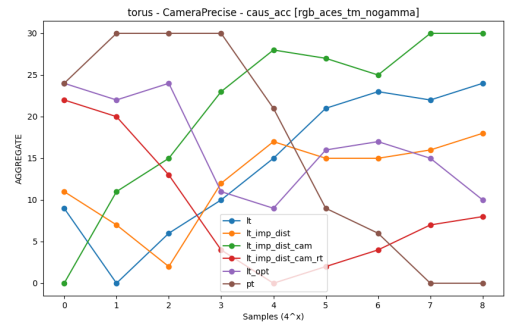
(c) DICE



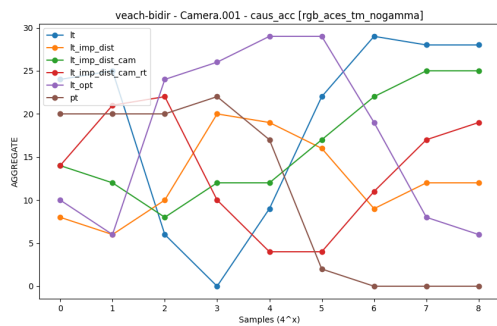
(d) REFLECT



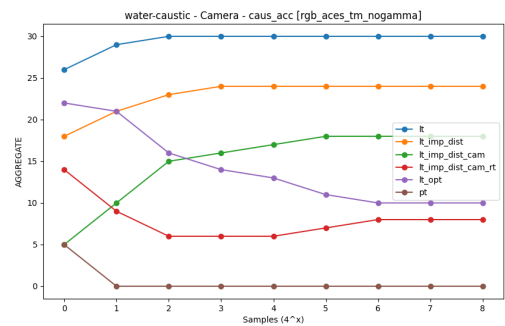
(e) OUTER



(f) TORUS

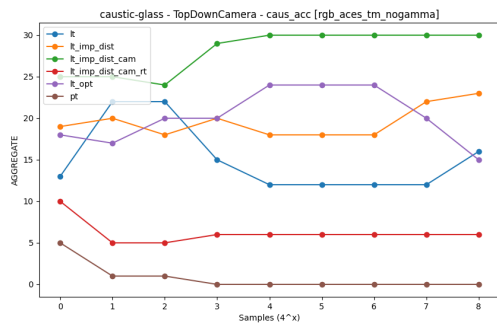


(g) EGG

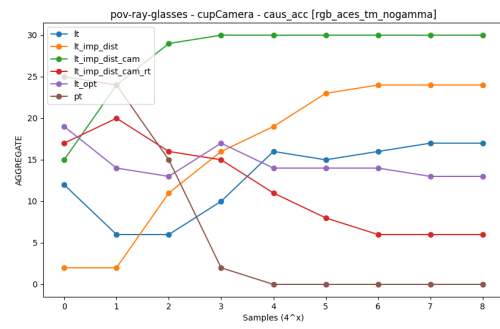


(h) WATER

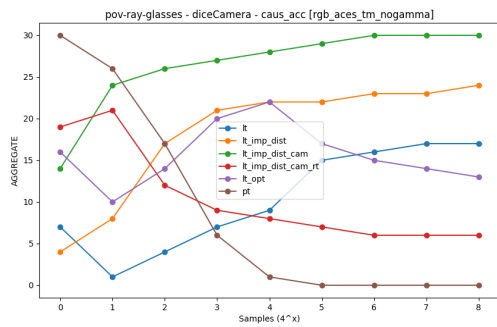
Figure 8.18 Aggregate



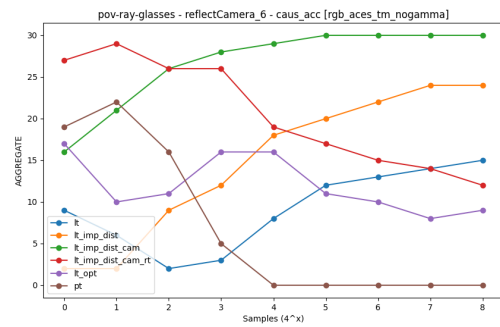
(a) CUP



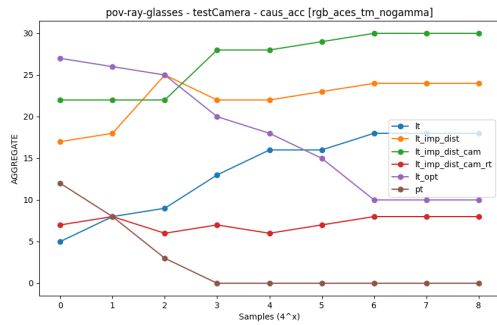
(b) CUPS



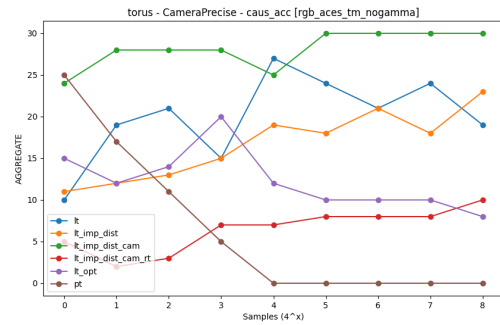
(c) DICE



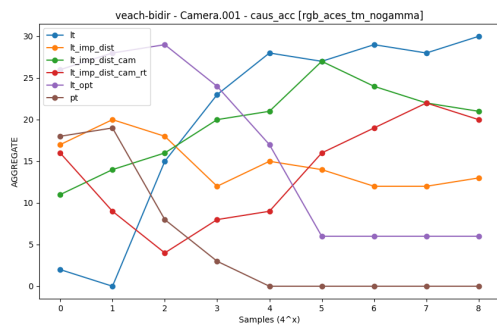
(d) REFLECT



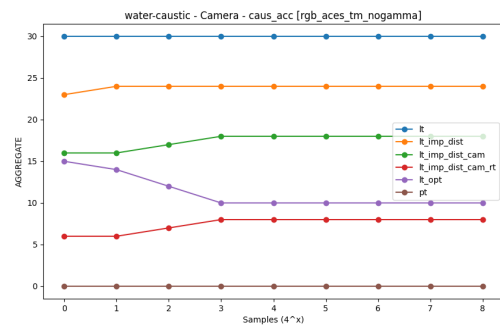
(e) OUTER



(f) TORUS

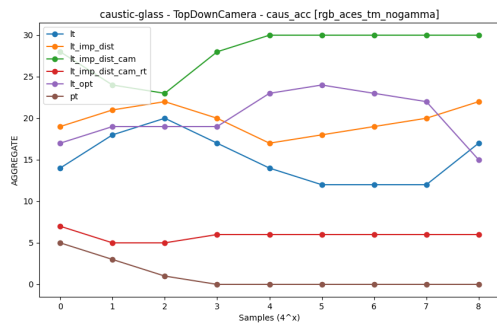


(g) EGG

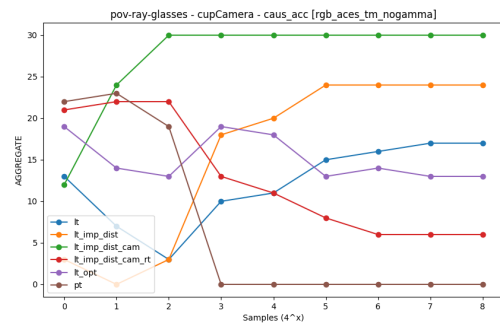


(h) WATER

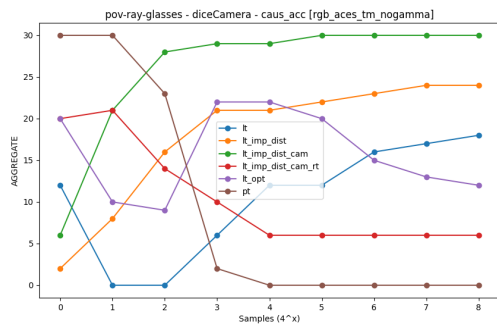
Figure 8.19 Aggregate - Blurred - 2



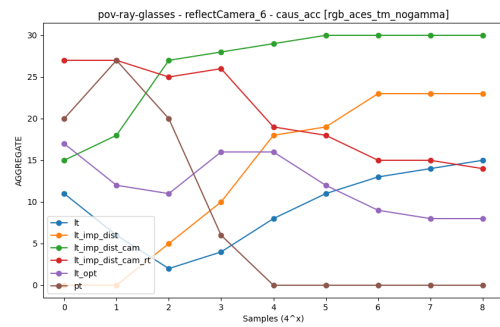
(a) CUP



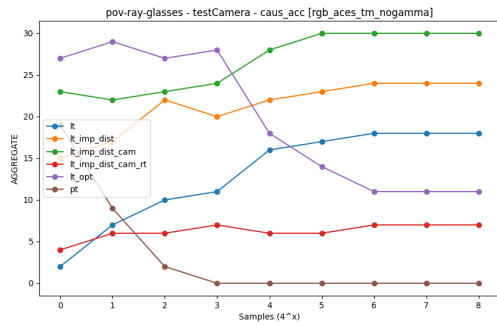
(b) CUPS



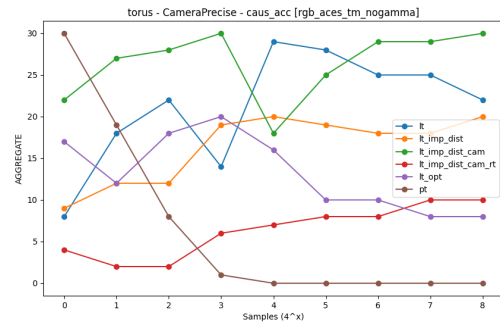
(c) DICE



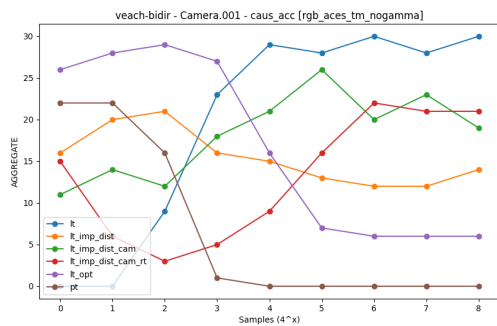
(d) REFLECT



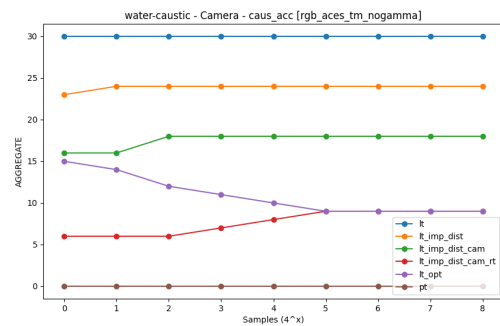
(e) OUTER



(f) TORUS

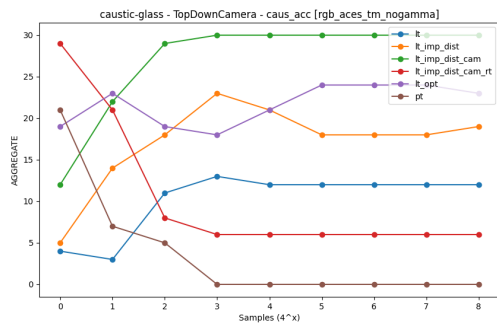


(g) EGG

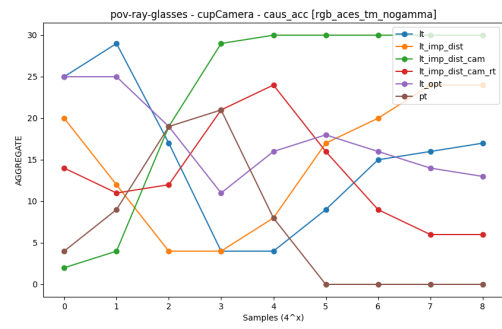


(h) WATER

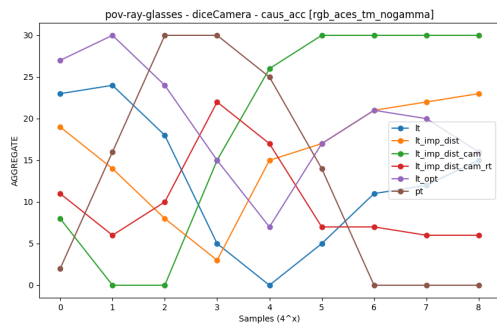
Figure 8.20 Aggregate - Downsampled 25%



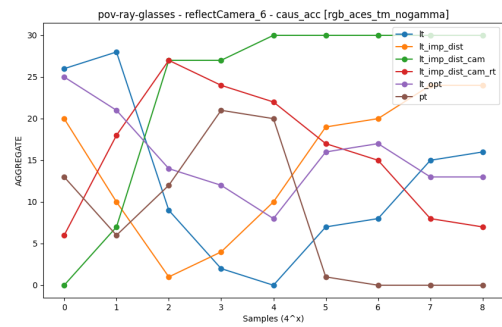
(a) CUP



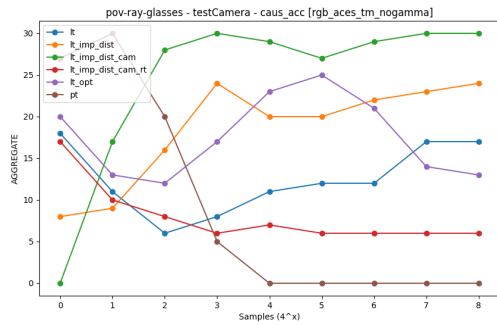
(b) CUPS



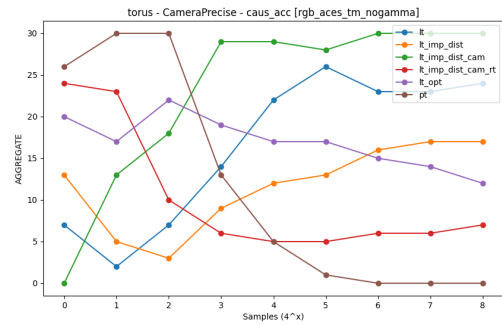
(c) DICE



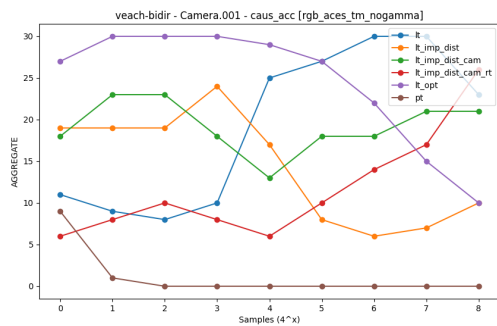
(d) REFLECT



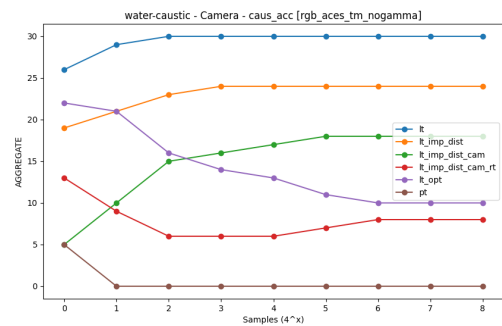
(e) OUTER



(f) TORUS



(g) EGG



(h) WATER

Figure 8.21 Aggregate - Cropped

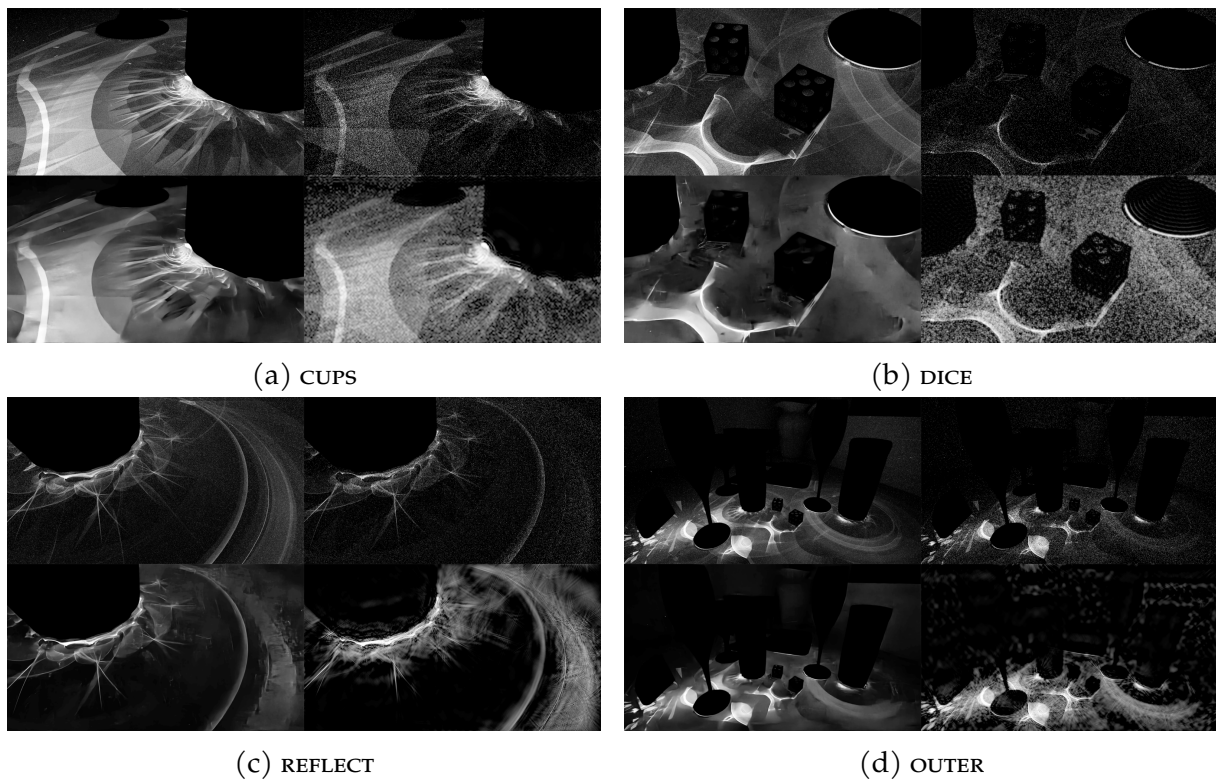


Figure 8.22 Collection of various images used in the study (image brightness enhanced for illustrative purposes). Each image is divided into four quadrants. Top-left: Ground truth image. Top-right: Noisy input. Bottom-left: Result from the OptiX denoiser. Bottom-right: Best denoised candidate from traditional denoisers.

8.6.2 Denoising

Table 8.23 presents the scores for each metric on each noisy image. Table 8.6 lists the ReLAX and ReBLUR scores for denoised images with and without caustic buffer accumulation. Table 8.7 details the parameters used to achieve the best scores across all metrics. Table 8.8 shows the best aggregate scores sorted using Algorithm 5, while Table 8.9 lists the parameters used to obtain these scores. Table 8.10 contains the scores for OptiX with caustic accumulation. Table 8.11 summarises the best scores across all metrics. Figure 8.24 compares the OptiX denoised images with and without importance and Figure 8.25 compares the denoised spectral images for the aggregate metrics with and without importance.

	FLIP	VDP3	MSE	PSNR	SSIM
s	1.64e-01	2.51	1.81e-02	17.41	0.42
D	1.26e-01	2.63	1.58e-02	18.01	0.35
R	3.77e-02	2.29	2.36e-03	26.28	0.68
O	5.44e-02	2.59	3.60e-03	24.44	0.66

Figure 8.23 Metric scores for the noisy images.

	FLIP	FLIP	VDP3	VDP3	MSE	MSE	PSNR	PSNR	SSIM	SSIM
	Acc.		Acc.		Acc.		Acc.		Acc.	
s	9.23e-02	8.35e-02	4.40	4.63	3.12e-03	2.48e-03	25.06	26.05	0.61	0.62
D	7.05e-02	6.98e-02	2.35	2.26	3.07e-03	3.17e-03	25.13	24.71	0.51	0.50
R	2.05e-02	1.99e-02	5.22	5.35	3.43e-04	3.29e-04	34.65	34.83	0.83	0.84
O	2.58e-02	2.64e-02	6.19	5.96	5.11e-04	6.22e-04	32.83	31.98	0.86	0.86
s	1.02e-01	1.15e-01	4.13	3.09	3.90e-03	5.94e-03	24.09	22.27	0.56	0.51
D	7.61e-02	8.49e-02	2.71	1.83	3.41e-03	5.13e-03	24.37	22.60	0.47	0.41
R	2.09e-02	2.13e-02	5.42	5.73	2.94e-04	2.73e-04	34.92	34.90	0.82	0.79
O	2.88e-02	3.29e-02	5.94	5.24	6.77e-04	9.73e-04	31.62	30.04	0.83	0.80

Table 8.6 Accumulating and non-accumulating scores. Best score for each scene-metric pair shown in bold. Top: ReLAX. Bottom: ReBLUR.

	Metric	Search	Denoiser	Thres.	$T_s(s)$	$T_d(s)$	Score
s	FLIP	powell	curvelet	garrote	982	3.27	8.46e-02
s	VDP3	powell	fourier	soft	39798	0.27	5.98
s	MSE	cobyla	curvelet	mult	395	3.15	2.48e-03
s	PSNR	powell	wavelet	hard	67	0.05	28.15
s	SSIM	powell	wavelet	hard	418	0.05	0.78
D	FLIP	powell	curvelet	soft	883	3.44	7.06e-02
D	VDP3	gp	fourier	hard	2232	0.28	5.07
D	MSE	powell	curvelet	hard	470	3.17	3.41e-03
D	PSNR	powell	wav/20	hard	105	0.18	27.90
D	SSIM	powell	wavelet	hard	511	0.05	0.70
R	FLIP	powell	curvelet	mult	2206	3.15	1.82e-02
R	VDP3	powell	curvelet	garrote	2925	3.21	7.44
R	MSE	powell	curvelet	garrote	411	3.21	1.39e-04
R	PSNR	trust	curvelet	garrote	1667	3.29	37.83
R	SSIM	powell	curvelet	mult	730	3.06	0.89
O	FLIP	powell	curvelet	soft	842	3.27	2.66e-02
O	VDP3	powell	curvelet	hard	5541	3.16	7.32
O	MSE	powell	curvelet	garrote	802	3.29	3.25e-04
O	PSNR	bfgs	curvelet	garrote	637	3.12	34.80
O	SSIM	powell	curvelet	mult	812	3.01	0.89

Table 8.7 Best scores from multiple runs (detailed); gp is Bayesian optimisation. T_s is search time, T_d is denoising time.¹

When comparing the optimal results in Table 8.11 with those of ReLAX, it is evident that REFLECT exhibited superior performance across all metrics. The CUP and

	FLIP	VDP3	MSE	PSNR	SSIM
s	9.24e-02	5.33	2.48e-03	26.07	0.67
d	8.40e-02	3.82	3.52e-03	24.24	0.55
r	2.19e-02	7.20	1.39e-04	37.83	0.87
o	2.79e-02	7.08	3.25e-04	34.80	0.87

Table 8.8 Aggregate metrics with all metrics computed.

	Metric	Search	Denoiser	Thres.	$T_s(s)$	$T_d(s)$	Score
s	MSE	cobyla	curvelet	mult	395	3.05	2.48e-03
d	MSE	cobyla	fourier	mult	138	0.29	3.52e-03
r	MSE	trust	curvelet	garrote	940	3.20	1.39e-04
o	MSE	bfgs	curvelet	garrote	485	3.25	3.25e-04

Table 8.9 Aggregate metrics. T_s is search time, T_d is denoising time.¹

DICE methods demonstrated improved results in HDRVDP3, PSNR, and SSIM, while OUTER performed well in all metrics except FLIP. In comparison with ReBLUR, all scenes showed enhanced performance across all metrics, with the exception of DICE, which exhibited comparable MSE values. Relative to OptiX, the majority of results were generally less favourable, though HDRVDP3 scores improved in all cases. The DICE method achieved higher PSNR values, while CUP and DICE performed particularly well in SSIM. As shown in Table 8.7, the curvelet transform generally delivered robust performance.

When examining the results presented in Chapter 7, the spectral denoisers in Table 8.11 demonstrate superior performance to those in Table 7.7 for the four tested scenes, with the exception of CUPS under the FLIP metric and DICE under the PSNR metric. Similarly, the spectral denoisers in Table 8.8 achieve better results than those in Table 7.4, except for CUPS when evaluated using the FLIP and PSNR metrics.

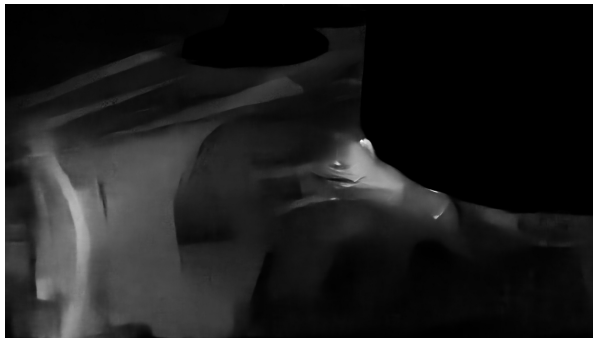
A comparison between the denoised outputs from ReLAX and ReBLUR in Table 8.6 and their non-importance-based counterparts in Table 7.2 reveals that generally the importance-based methods yield higher scores, except for CUPS under the FLIP metric in both ReLAX and ReBLUR. In the case of OptiX, all scenes denoised from importance-based noisy inputs in Table 8.10 consistently outperform those in Table 7.6 across every metric.

8.7 Discussion

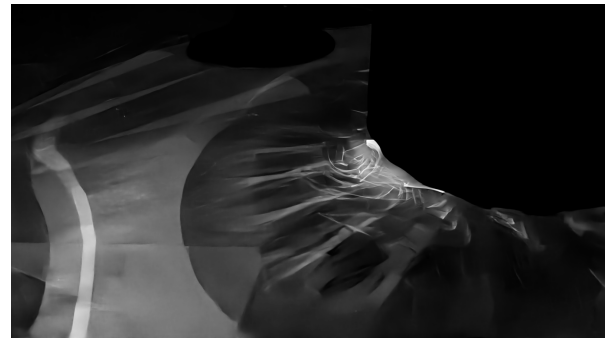
When compared with other shaders, LT_IMP_DIST_CAM outperforms others in 75% of cases at high spp. However, according to the rendered images at lower spp, the initial results seem to be off. The problem is that the path tracer achieves a very high score in scenes such as DICE. As outlined earlier, this should not be possible since, for this scene, the path tracer does not output any direct caustics (see Figure 8.11).

Initially, we considered whether the problem might relate to the fact that the caustics occupy a small area relative to the buffer size. New comparisons were made using cropped buffers (see Figure 8.21 and Figure 8.4h), ensuring that only regions where caustics are the dominant feature were retained. However, the results did not

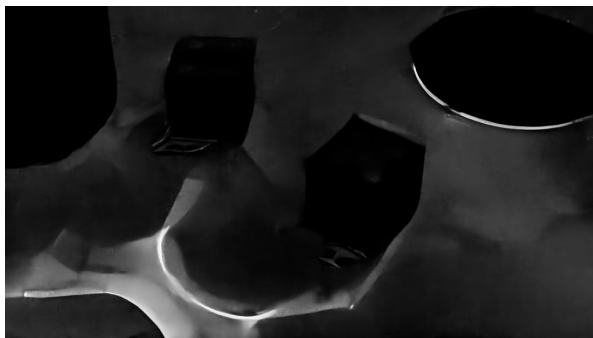
¹Search runs distributed on AMD Ryzen 3600X, 3960X, 7950X, Intel i7-8700K (a) + NVIDIA RTX 2080 Ti (b). Only FLIP on (b). Denoising on (a).



(a) CUPS



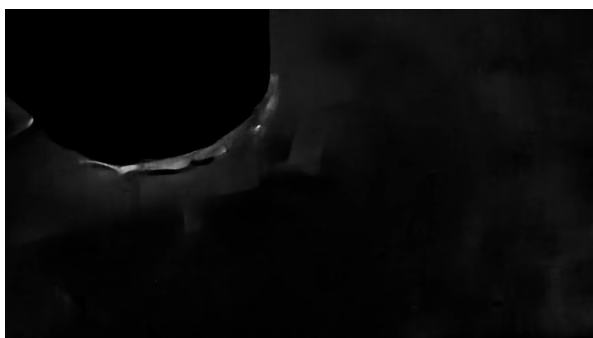
(b) CUPS (IMPORTANCE)



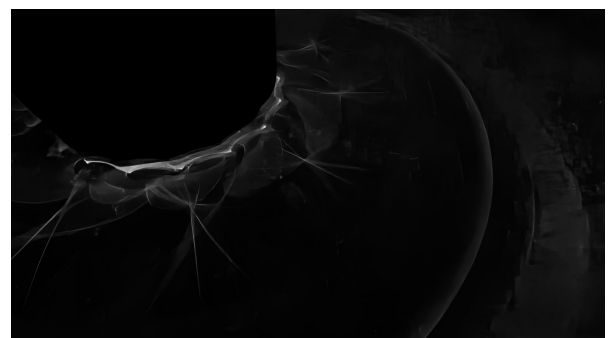
(c) DICE



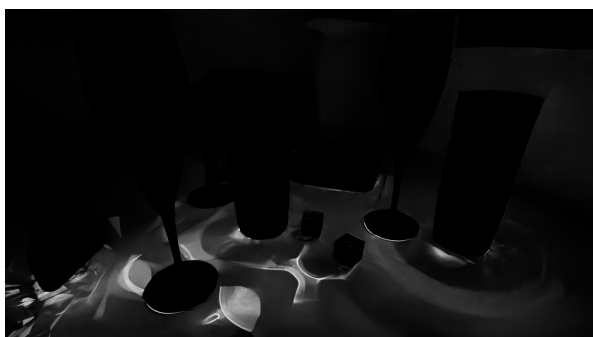
(d) DICE (IMPORTANCE)



(e) REFLECT



(f) REFLECT (IMPORTANCE)

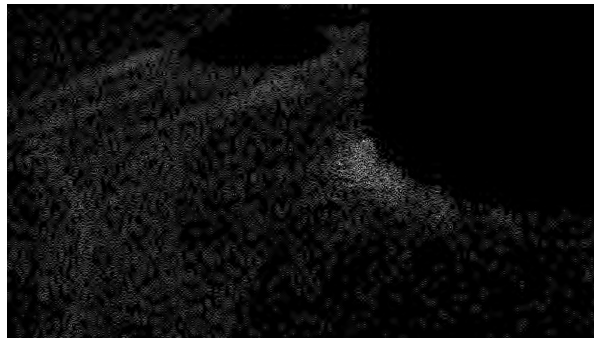


(g) OUTER

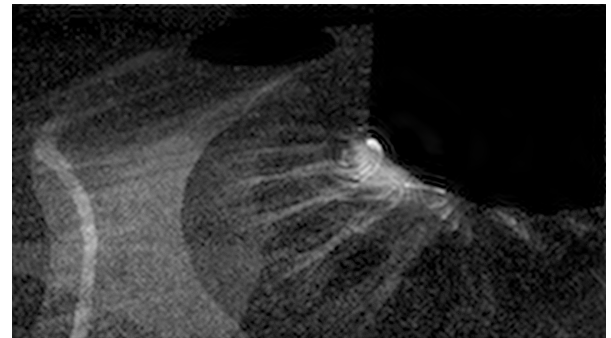


(h) OUTER (IMPORTANCE)

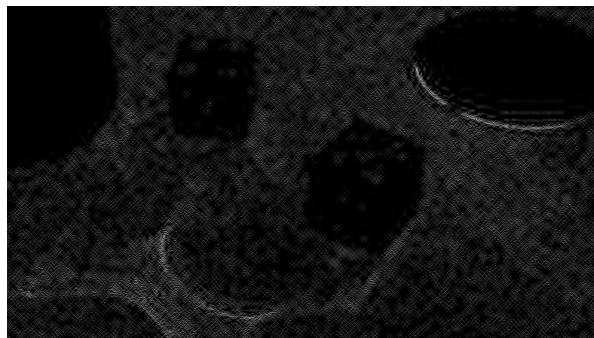
Figure 8.24 OptiX denoised. Non-importance vs. importance.



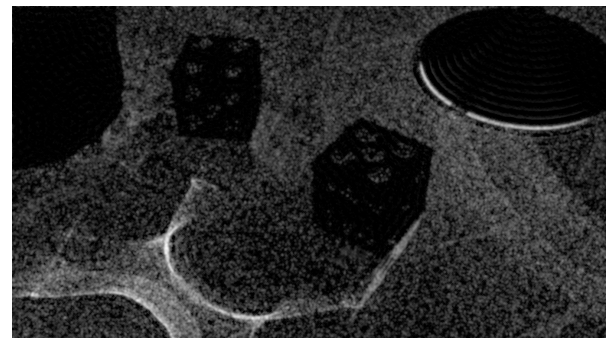
(a) CUPS



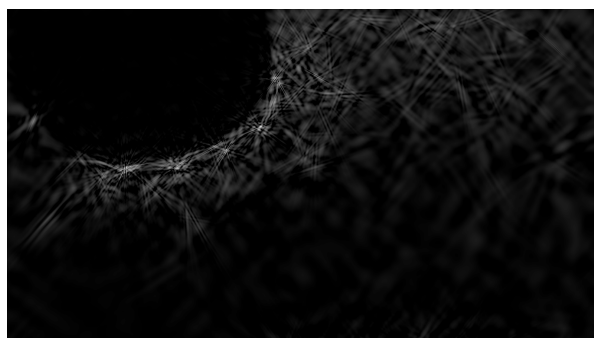
(b) CUPS (IMPORTANCE)



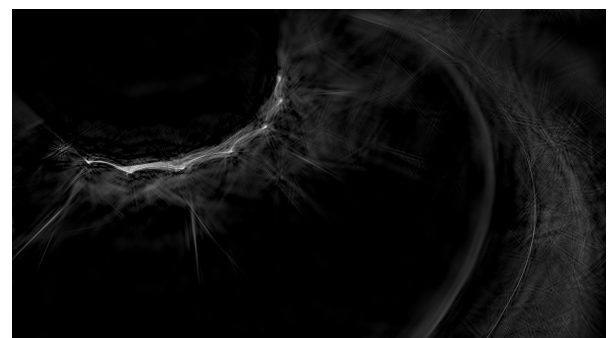
(c) DICE



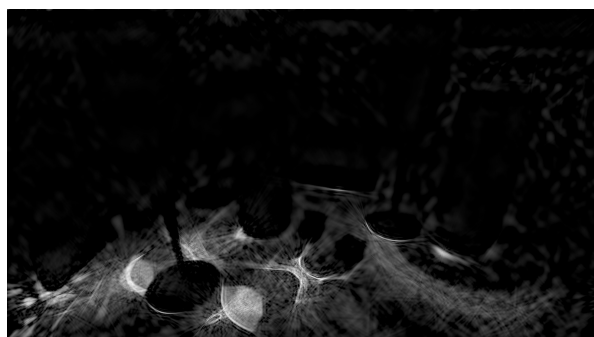
(d) DICE (IMPORTANCE)



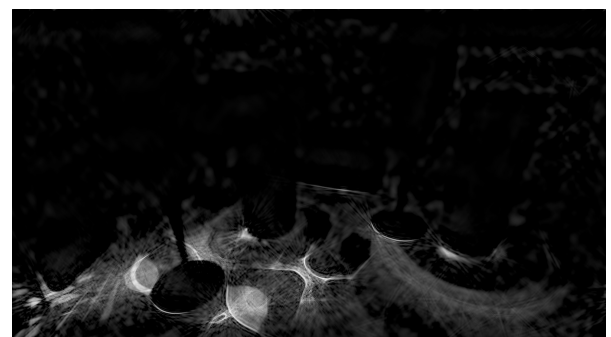
(e) REFLECT



(f) REFLECT (IMPORTANCE)



(g) OUTER



(h) OUTER (IMPORTANCE)

Figure 8.25 Spectral denoised. Non-importance vs. importance.

	TLIP	VDP3	MSE	PSNR	SSIM
S	6.30e-02	4.17	9.52e-04	30.21	0.73
D	5.95e-02	2.42	1.68e-03	27.45	0.61
R	1.69e-02	6.92	1.24e-04	38.33	0.90
O	2.05e-02	6.71	2.24e-04	36.41	0.91

	TLIP	VDP3	MSE	PSNR	SSIM
S	8.46e-02	5.98	2.48e-03	28.15	0.78
D	7.06e-02	5.07	3.41e-03	27.90	0.70
R	1.82e-02	7.44	1.39e-04	37.83	0.89
O	2.66e-02	7.32	3.25e-04	34.80	0.89

Table 8.10 OptiX accumulating scores. Table 8.11 Best scores from multiple runs.

show significant improvement, with the path tracer still scoring highly in DICE. This indicates that the problem lies elsewhere, and we suspected that noise might be the issue.

To test whether noise is the primary problem, we blurred all images and computed new scores (see Figure 8.19 and Figure 8.4f). Blurring serves as a basic low-pass filter, attenuating high frequency noise that is prominent at lower spp. While blurring affects the caustic pattern, at lower sample counts it produces a better quality image. With this method, the path tracer declines sharply at higher sample counts, as caustics begin to form with the other shaders. Downsampling has a similar effect to blurring, as noise is reduced because a pixel in the downsampled version effectively integrates more samples than in the original, depending on the scale used for downsampling.

For most scenes, `LT_IMP_DIST_CAM` performed better. This technique assigns importance to distant objects and to objects that are closer to the viewing camera, which helps redistribute light rays to where they are needed. It performs especially well in scenes with a high number of specular objects, such as `CUP`, `DICE`, `REFLECT`, and `OUTER`. The caustics in these scenes are superior to those produced by any other shader for the same number of samples. `LT_OPT` performs poorly in these scenes, as assigning importance to specular objects when the scene is densely populated with them becomes as ineffective as random sampling. However, in scenes with fewer specular objects such as `EGG`, `LT_OPT` performs best. In `EGG`, ring pattern caustics beneath the glass egg are already visible. This effect is not achieved by other shaders with the same number of samples. In `WATER`, `LT` performed best. In the renders, the caustics on the ceiling are visibly less noisy than those from any other shader.

`LT_IMP_DIST_CAM_RT` has been designed for indirect caustics, prioritising light rays that are visible to the camera. The tested scenes reveal that parts of the caustic pattern are missing. For instance, in `DICE`, the curves at the centre of the caustics are absent. This occurs because the large cup on the right occludes direct visibility from the floor to the light, causing the importance map to cull this region.

In most cases, `LT_IMP_DIST` performed better than `LT`. However, `LT_IMP_DIST` is never the best performing shader in any scene or at any sample count.

8.7.1 Denoising

Overall, the comparative performance observed among spectral denoisers, ReBLUR, ReLAX and OptiX in these results aligns with the findings presented in Chapter 7. Specifically, ReBLUR and ReLAX exhibited inferior performance relative to spectral denoisers, which themselves demonstrated weaker results compared to OptiX.

`cups` demonstrates significantly improved caustic pattern definition when assessed via `FLIP`, attributable to the curvelet transform’s effective preservation of edge characteristics and directional information. Conversely, ReLAX-produced images exhibit noticeable blurring and increased noise levels. OptiX consistently outperforms other methods, delivering exceptionally sharp caustics and noise-free results across all test scenes. Nevertheless, conventional denoising approaches occasionally achieve superior outcomes – particularly in the `OUTER` scene (Figure 8.27), where some curvilinear features within caustic patterns become attenuated. In these specific cases, curvelet-based methods retain their performance advantage.

The performance of the metrics used to determine denoising coefficients varied significantly. Overall, `FLIP` performed well, except in `REFLECT` where the scene appeared overexposed, resulting in a loss of detail. Conversely, the `HDRVDP3` metric still yielded suboptimal results due to the presence of clustered speckle noise, as illustrated in Figure 8.26, despite the input being less noisy. `MSE` still performed well, except in the `cups` scenario, where the image appeared blurred. Again, `PSNR` underperformed in scenes such as `DICE` and `cups`, despite its close relationship with `MSE`. Similarly, `SSIM` was the least effective metric, producing unsatisfactory results in the `cups` and `DICE` scenarios, where pixelation artefacts were evident. Additionally, in the `OUTER` and `REFLECT` scenarios, the images appeared overexposed, making the caustic patterns barely discernible.

The enhanced caustic buffers, when processed with noisy importance-based inputs, exhibited significant improvements. As illustrated in Figure 8.24 for OptiX, `cups` and `REFLECT` now display intricate caustic patterns that were previously indiscernible, having been replaced by indistinct blurred regions of light. Within `DICE`, additional curvilinear features emerge along the caustic’s periphery, particularly near the physical dice. The `OUTER` scene demonstrates more nuanced refinements owing to the diminutive scale of its caustics. Notably, the caustic formations exhibit greater definition, and when examining the diffusely reflective vase in the background, alterations in shading illumination attributable to caustics become apparent.

The spectral denoising results presented in Figure 8.25 demonstrate enhanced quality in the caustics buffer when processing importance-based noisy inputs. Unlike the OptiX case, the non-importance images do not exhibit blurring but rather display curve-like artefacts introduced by the curvelet transform. When denoising importance-based inputs, these artefacts diminish, thereby allowing the underlying caustic patterns

to emerge. Mirroring the OptiX observations, substantial enhancements are evident in CUPS and REFLECT, where intricate details near glass surfaces become clearly visible, while DICE and OUTER exhibit more subtle yet discernible improvements.

Although the figures have been enhanced in brightness to improve visibility of changes (thereby accentuating the curve-like artefacts), OptiX maintains superior denoising performance. While it exhibits a tendency to blur certain details and may lose some caustic information during processing, its output remains virtually noise-free. This approach strikes an effective compromise between preserving structural details and suppressing noise.

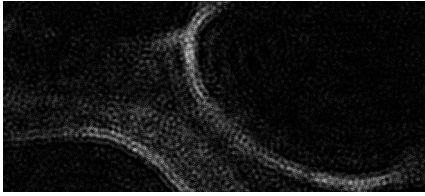


Figure 8.26 Clustered speckle noise in DICE with VDP3.

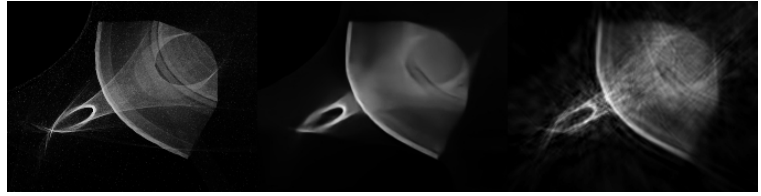


Figure 8.27 Left: Reference. Middle: OptiX. Right: Curvelet.

8.8 Summary

This chapter explores enhancements to the CLT to achieve lower noise in unbiased rendering at 16 spp, addressing the limitations of spectral denoising and AI-based denoisers like OptiX in preserving caustic details. Building on prior findings, the chapter introduces two importance sampling techniques: COLT and Importance Maps. COLT mimics NEE by strategically sampling specular materials, proving especially effective in scenes with few reflective objects. Importance Maps, implemented as 2D probability textures, guide light sampling using three methods; distance-based weighting, camera-aware weighting (incorporating view direction and occlusion), and ray-traced visibility culling. These are used to prioritise light paths contributing most to the final image.

The evaluation leverages scenes from earlier denoising studies, employing metrics like MSE, SSIM, PSNR, HDRVDP3, FLIP, and LPIPS, alongside a revised aggregate metric. Results demonstrate that camera-aware importance maps (`LT_IMP_DIST_CAM`) consistently outperform other methods at higher spp by efficiently redistributing light to visible, distant regions, while COLT (`LT_OPT`) excels in scenes with sparse specular objects (e.g., egg). Notably, path tracing scores deceptively well at low spp due to metric sensitivity to noise, but this artefact vanishes when images are blurred or down-sampled, confirming PT's inadequacy for caustics.

Denoising experiments reveal that importance-based inputs improve all denoisers, with OptiX remaining superior but spectral methods closing the gap in preserv-

ing structural details (e.g., curvelets outperform OptiX in retaining caustic curves in OUTER). However, limitations persist: Importance Maps are costly for dynamic scenes or multi-light setups, and metric inconsistencies (e.g., SSIM's poor performance with pixelation artefacts) underscore challenges in low-spp evaluation. The chapter concludes that `LT_IMP_DIST_CAM` is ideal for high-quality rendering, while `LT_OPT` or standard `LT` may suffice for specific low-spp or low-specular scenarios, emphasising the need for noise-robust metrics in real-time applications.

9 Conclusion

The unbiased depiction of intricate light phenomena, including caustics, can be obtained through offline Monte Carlo ray tracing methods such as MLT and BPT. In real-time rendering, however, techniques for caustics have largely relied on approximations, employing either screen-space approaches that lack accuracy or precomputed light paths that sacrifice dynamism, owing to the stringent computational demands and limited time constraints. The advent of hardware-accelerated ray tracing in contemporary GPUs has gradually enabled the adoption of Monte Carlo-based ray tracing techniques in real-time applications. Nevertheless, these methods predominantly address more straightforward global illumination effects, such as colour bleeding and direct reflections. Rendering caustics in real time remains particularly challenging due to their complex light transport paths and the need for precise angular sampling, which often results in excessive noise under tight temporal constraints.

The primary aim of this thesis has been to develop methods for rendering caustics in real time. Our approach involved surveying the extensive body of offline rendering literature and adapting suitable techniques for implementation on GPU hardware. Due to architectural differences, we established key criteria to accommodate GPU limitations, such as avoiding complex data structures and prioritising texture-based storage where necessary. Path tracing emerged as the most straightforward method compatible with GPU constraints while still capable of rendering caustics.

To ensure methodological correctness, Chapter 4 describes our initial implementation of a CPU-based path tracer with a streamlined yet robust design, facilitating later GPU adaptation and verification of techniques. Prior to deployment on the GPU, we developed auxiliary debugging tools. During GPU implementation, issues such as fireflies and random black pixels arose. Recognising that debugging on both CPU (visualisation challenges) and GPU (lack of straightforward diagnostic tools) posed significant hurdles, particularly for more complex algorithms, we consolidated these tools into a unified framework named Anvil. This framework enables real-time visualisation of common data structures like rays and supports breakpoint functionality, vastly accelerating debugging and development.

Chapter 5 introduces light tracing as our second candidate for GPU implementation. Its inherently stateless nature makes it well-suited to GPU architectures, and its light-based path generation outperforms path tracing in rendering caustics. We first conducted a feasibility study using Anvil to determine whether CPU-based light tracing could generate visible caustics within a few frames. Based on these promising results, we proceeded with the GPU implementation, documenting all encountered technical challenges. We conducted evaluations across multiple benchmark scenes, confirming

that the GPU-based light tracer operates within real-time constraints while meeting convergence metrics. However, the resulting caustics exhibited excessive noise.

To preserve the structural patterns in caustics while mitigating noise, we explored spectral denoising, a process divided into offline threshold coefficient determination and online application. Given the high-dimensional parameter space involved, Chapter 6 presents Forge, an extension of Anvil designed for distributed evaluation. Forge employs the same data model (atoms and molecules) while enabling large-scale, reproducible testing without modifying the evaluated software, thus reducing human error and configuration inconsistencies.

Chapter 7 details Forge-assisted spectral denoising experiments. After assessing multiple spectral algorithms, search methods, and thresholding techniques, with results validated within strict confidence intervals, we identified curvelets as the optimal solution, excluding the AI-based OptiX denoiser. Nonetheless, the denoised output occasionally exhibited artefacts or excessive blurring.

Chapter 8 examines importance-based sampling to reduce noise in light-traced caustics at fixed sample counts. The refined signal undergoes denoising once more, ultimately yielding higher-fidelity caustics.

9.1 Contributions

The primary contribution of this thesis lies in the unbiased real-time generation of caustics. We introduce an importance-driven light tracing algorithm that generates caustics with noise levels demonstrably suitable for subsequent denoising processes. The remaining sections outline the specific contributions organised by chapter.

9.1.1 Anvil

Anvil represents a visual debugging framework specifically developed to address the challenges inherent in physically based rendering, particularly stochastic variations like Monte Carlo rendering. Utilising the ECS architectural pattern, it provides scalable low-level primitives well-suited for computer graphics applications. By employing atom and molecule mapping, the tool enables users to visually inspect ray propagation patterns, energy transfer quantities, and related phenomena. The system’s modular architecture allows for straightforward integration of additional visualisers to address previously unexplored debugging scenarios. Its development was driven by practical demands, aiming to enhance existing debugging methodologies. Empirical case studies demonstrate that Anvil significantly enhances productivity by accelerating the process of identifying and resolving rendering issues.

List of Contributions

- A versatile debugging and visualisation framework specifically tailored for stochastic rendering applications
- Supports interactive debugging capabilities enabling exploratory analysis during development
- Features modular architecture permitting effortless integration and reuse of visualisation components
- Demonstrates sufficient abstraction for potential adaptation to other technical domains

9.1.2 Light Tracing

The GPU-based implementation of light tracing was chosen to accelerate the convergence of real-time caustics due to its unbiased nature and stateless characteristics. This approach offers distinct advantages over alternative techniques such as PM or BPT. The system permits adjustable bit-depth settings for irradiance accumulation and segregates radiance into multiple buffers to facilitate analysis and potential denoising during the research process. Rigorous testing was conducted using Anvil, with comparisons drawn against both path tracing and light tracing outputs. Experimental results demonstrate a mean rendering duration of 3.27 milliseconds per frame, confirming adherence to real-time performance requirements. This foundational work will support the development of importance-based light tracing in subsequent chapters.

List of Contributions

- A real-time GPU implementation of unbiased light tracing methodology
- Output generation designed for seamless integration with contemporary denoising systems

9.1.3 Forge

Forge represents an extension of the Anvil framework, specifically developed to standardise and optimise assessment procedures within computer graphics pipelines. This system addresses multiple critical challenges in the field, including maintaining configuration consistency across diverse renderers, reducing potential human errors, and improving the repeatability of experimental outcomes. Through these advancements,

Forge offers researchers and developers a dependable platform for conducting precise comparative analyses of rendering methodologies.

The framework serves as the foundation for evaluation in spectral denoising, particularly in determining optimal thresholding coefficients. Building upon Anvil’s existing capabilities, Forge introduces support for distributed evaluations, enabling execution both on local systems and across networked environments. This flexibility facilitates integration with existing functionality from libraries implemented in various programming languages. For instance, in our work, image similarity metrics such as HDRVDP3 were accessed via MATLAB, while FLIP and other metrics were computed using Python, with thresholding coefficient searches conducted efficiently across these environments. Additionally, Forge was employed to verify the accuracy of the GPU-based light tracing implementation discussed in Chapter 5, further ensuring the reliability of our results.

List of Contributions

- A universal evaluation framework specifically developed for stochastic rendering applications
- Builds upon the Anvil architecture, facilitating the reuse of fundamental rendering components (atoms/molecules)
- Supports decentralised evaluation with real-time synchronisation capabilities
- Employs WebSocket-based synchronisation enabling globally distributed component integration via internet infrastructure

9.1.4 Spectral Denoising

The high noise levels in our light tracer implementation prompted an evaluation of spectral transforms (Fourier, wavelet, and curvelet) for denoising caustics in Monte Carlo rendering, where we employed Forge to determine optimal filtering coefficients in the spectral domain. While curvelet transforms proved superior to traditional methods in preserving high-frequency details, they were ultimately outperformed in overall noise reduction by AI-based denoisers like OptiX. Despite this limitation, our investigation demonstrated promising results regarding coefficient reuse across colour signals, different scene viewpoints, and entirely distinct scenes, suggesting potential broader applications for these spectral techniques. The effectiveness of spectral denoising is additionally assessed when applied to importance-based light tracing, demonstrating that even marginal improvements in input quality yield measurable enhancements in the final denoised results.

List of Contributions

- A systematic comparison of traditional image transforms against state-of-the-art denoising techniques in the context of denoising caustics from inputs with low sample counts.
- A method for searching for the optimal settings of coefficients for Fourier, wavelet and curvelet transforms, tailored to the specific challenge of caustics noise reduction, using statistical methods to validate these findings.
- The use of a comprehensive set of evaluation metrics to assess image quality, offering insights into the strengths and limitations of each metric when applied to the problem of denoising caustics in PBR-generated images.

9.1.5 Importance

In an attempt to reduce the noise produced by the GPU-based light tracer without increasing the sample rate (16 spp), this chapter explored the implementation and evaluation of GPU-based light tracing techniques for rendering caustics in real-time, with a focus on the CandelaDXR renderer built using DXR. Several variations of the light tracer were introduced, including the Naive Light Tracer, Caustics-Optimised Light Tracer, and Importance Map Light Tracer, each addressing specific challenges in caustic rendering. The results highlighted the strengths of the Importance Map Light Tracer, particularly the `LT_IMP_DIST_CAM` shader, which outperformed other methods in most scenes by effectively redistributing light rays to areas of interest. Evaluation of the enhanced light tracer with spectral denoising revealed substantial improvements in certain scenes, including cases where the curvelet-based approach better preserved fine details compared to the OptiX denoiser.

List of Contributions

- COLT - a GPU-optimised light tracing framework specifically designed for caustics, featuring direct sampling of specular surfaces
- An importance-driven GPU light tracer incorporating camera position, distance metrics and occlusion data to inform light path generation
- A comprehensive assessment of spectral denoising techniques using enhanced input data sources

9.2 Findings and Insight

This research presents novel developments in real-time caustics rendering. It proposes a visual debugging methodology employing established computer graphics constructs (including rays and radiance) to facilitate the transition of offline Monte Carlo techniques into real-time applications. Conventional debugging tools like NVIDIA Nsight and Pix primarily focus on binary-level analysis, proving inadequate for visualising typical rendering issues, particularly those stemming from erroneous probability distributions. By utilising Anvil, we achieved operation at a higher-level of abstraction level, thereby exposing previously obscured rendering artefacts.

When adapting the light tracing algorithm, the complexities inherent in GPU implementation became apparent. The challenges associated with the PRNG employed in the process proved particularly difficult to identify. Nevertheless, light tracing offers an efficient implementation that maximises the potential of GPU hardware. The algorithm's performance exhibits a direct correlation with the number of samples. In contrast to path tracing, sample rates remain independent of output resolution, allowing for the application of adaptive techniques to optimise the trade-off between noise reduction and computational efficiency.

Importance-based approaches yield noticeable enhancements with minimal computational overhead in particular scenarios. Measuring the extent of improvement, however, presented difficulties. The evaluation indicated that none of the similarity assessment methods tested reliably reflected the qualitative observations. Images perceived as superior were conversely rated as inferior, and vice versa. Experimental findings suggested that reducing noise restored the dependability of these metrics. Subsequent testing supported this proposition, demonstrating that assessments of blurred images aligned with expectations. This correlation is logical, as reduced noise in blurred images enhances the consistency of similarity evaluations.

Spectral denoising demonstrated competitive image quality relative to ReLAX and ReBLUR, while matching OptiX in only select instances. OptiX exhibits remarkable performance in terms of both speed and visual fidelity, even with limited samples. Nevertheless, it fails to accurately retain all caustic details and often introduces excessive blurring. A combined methodology, such as integrating curvelet outputs into OptiX, holds promise for enhanced denoising quality. It should be noted that only a subset of variable coefficients was chosen for thresholding to minimise computational overhead during both search and denoising operations. This restriction may inherently constrain the denoising efficacy.

9.3 Limitations and Future Work

Anvil significantly improves debugging workflows for physically based rendering, but several constraints remain. The reflection system currently enforces read-only access to debugged data structures, requiring code recompilation for parameter adjustments that could otherwise be modified interactively. Integration challenges emerge when incorporating Anvil into existing codebases, particularly those sharing dependencies like ImGui, potentially causing rendering artefacts or race conditions without careful state management. However, its modular design permits community-driven extensions that could address these limitations.

Regarding rendering techniques, naive light tracing exhibits substantial noise at low sample counts, as corroborated through both HDRVDP3 metrics and visual inspection. The importance-based variants face scalability challenges, requiring complete map regeneration for camera movements or light transformations. While single-map configurations with random light selection provide unbiased multi-frame results, they introduce per-frame bias. The Forge framework presents additional limitations, including a steep learning curve for new users and potential network latency when using its WebSocket system for distributed evaluations.

Future development priorities for Anvil include implementing write-back functionality in the reflection system and automating GPU-host data transfers through specialised macros. For Forge, enhanced documentation and example projects would improve accessibility, while WebSocket optimisations could reduce evaluation latency. Spectral denoising requires further investigation into coefficient reuse across scenes with similar frequency characteristics, along with GPU acceleration for real-time applications. Preliminary tests suggest that a small subset of coefficients may maintain quality while reducing computational costs.

Additional research directions include exploring spherical harmonics for efficient importance map storage and investigating hybrid rendering approaches that combine light tracing with path tracing. The implementation of stochastic strategy selection in bidirectional path tracing could enable dynamic, context-aware sampling. These advancements would address current limitations while expanding the practical applications of real-time caustic rendering in interactive graphics and VR environments.

9.4 Final Remarks

The real-time physically based rendering of caustics represents a significant challenge, the resolution of which would elevate the realism of interactive graphical applications, including video games and VR. Objects such as wine glasses, windows, and water

waves are currently rendered using approximations or precomputed techniques, limiting the fidelity of light interactions experienced by users in VR environments. This thesis advances the field by introducing Candela Path Tracing and Candela Light Tracing, methodologies adapted from offline rendering literature to accommodate the constraints of real-time performance and GPU architectures. Importance-based light tracing variants enable real-time caustic rendering at a quality suitable for subsequent denoising or post-processing. Investigations into specular denoisers identified curvelets as the most effective at preserving caustic detail. To facilitate debugging and ensure accuracy while minimising human error, the tools Anvil and Forge were developed and made publicly available, providing a firm foundation on which future research can build.

References

- [1] A. Harvill and P. A. Studios, "Effective toon-style rendering control using scalar fields,," in *SIGGRAPH Sketches*, 2007, p. 53.
- [2] J. T. Kajiya, "The rendering equation," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '86, New York, NY, USA: Association for Computing Machinery, 1986, pp. 143–150.
- [3] E. P. Lafortune and Y. D. Willems, "Bi-directional path tracing," in *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, Alvor, Portugal, Dec. 1993, pp. 145–153.
- [4] E. Veach and L. J. Guibas, "Metropolis light transport," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 1997, pp. 65–76.
- [5] G. W. Petty, *A first course in atmospheric radiation*. Sundog Publishing, 2006.
- [6] R. Montes and C. Ureña, "An overview of brdf models," *University of Grenada, Technical Report LSI-2012-001*, 2012.
- [7] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*, 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016, ISBN: 0128006455, 9780128006450.
- [8] B. T. Phong, "Illumination for computer generated pictures," *Communications of the ACM*, vol. 18, no. 6, pp. 311–317, 1975.
- [9] E. P. Lafortune and Y. D. Willems, *Using the modified phong reflectance model for physically based rendering*. Katholieke Universiteit Leuven. Departement Computerwetenschappen, 1994, vol. 197.
- [10] J. F. Blinn, "Models of light reflection for computer synthesized pictures," in *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, 1977, pp. 192–198.
- [11] R. L. Cook and K. E. Torrance, "A reflectance model for computer graphics," *ACM Transactions on Graphics (ToG)*, vol. 1, no. 1, pp. 7–24, 1982.
- [12] M. Oren and S. K. Nayar, "Generalization of lambert's reflectance model," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994, pp. 239–246.
- [13] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, "Microfacet models for refraction through rough surfaces,," *Rendering techniques*, vol. 2007, 18th, 2007.

- [14] K. Napoli, *Candela: A cpu-based renderer*, version v1.0, A cross-platform C++ rendering engine featuring multithreaded rendering, tiled rendering, and variance-based rendering., 2019. [Online]. Available: <https://bitbucket.org/kvnnap/candela>.
- [15] Unity Technologies. "Introduction to the data-oriented technology stack for advanced unity developers." [Online]. Available: <https://unity.com/resources/introduction-to-dots-ebook>.
- [16] S. Bilas, "A data-driven game object system," in *Game Developers Conference Proceedings*, 2002.
- [17] A. Martin. "Entity systems are the future of mmog development," Accessed: Jun. 1, 2023. [Online]. Available: <http://t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/>.
- [18] C. Szyperski, "Independently extensible systems-software engineering potential and challenges," *Australian Computer Science Communications*, vol. 18, pp. 203–212, 1996.
- [19] A. H. Eden and T. Mens, "Measuring software flexibility," *IEE Proceedings-Software*, vol. 153, no. 3, pp. 113–125, 2006.
- [20] I. Hickson and D. Hyatt, "Html5," W3C, Working Draft, 2008. [Online]. Available: <https://www.w3.org/TR/2008/WD-html5-20080122/>.
- [21] I. Fette and A. Melnikov, "The websocket protocol," IETF, RFC 6455, 2011. [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc6455>.
- [22] A. Russell, "Comet: Low latency data for the browser," *Alex Russell's Blog*, 2006. [Online]. Available: <https://infrequently.org/2006/03/comet-low-latency-data-for-the-browser/>.
- [23] V. Wang, F. Salim, and P. Moskovits, *The Definitive Guide to HTML5 WebSocket*. Apress, 2013.
- [24] A. Haar, "Application of the discrete wavelet transform to the monitoring of tool failure in end milling using the spindle motor current," *International Journal of Advanced Manufacturing Technology*, vol. 69, no. 3, pp. 331–371, 1910.
- [25] I. Daubechies, "The wavelet transform, time-frequency localization and signal analysis," *IEEE transactions on information theory*, vol. 36, no. 5, pp. 961–1005, 1990.
- [26] E. J. Candes, D. L. Donoho, et al., *Curvelets: A surprisingly effective nonadaptive representation for objects with edges*. Department of Statistics, Stanford University Stanford, 1999.

- [27] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.
- [28] F. Gao and L. Han, "Implementing the nelder-mead simplex algorithm with adaptive parameters," *Computational Optimization and Applications*, vol. 51, no. 1, pp. 259–277, 2012.
- [29] M. J. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The computer journal*, vol. 7, no. 2, pp. 155–162, 1964.
- [30] M. J. Powell, *A direct search optimization method that models the objective and constraint functions by linear interpolation*. Springer, 1994.
- [31] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on mathematical software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.
- [32] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SISC*, vol. 16, no. 5, pp. 1190–1208, 1995.
- [33] S. G. Nash, "Newton-type minimization via the lanczos method," *SIAM Journal on Numerical Analysis*, vol. 21, no. 4, pp. 770–788, 1984.
- [34] D. Kraft, "A software package for sequential quadratic programming," *Forschungsbericht-Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt*, 1988.
- [35] M. Lalee, J. Nocedal, and T. Plantenga, "On the implementation of an algorithm for large-scale equality constrained optimization," *SIAM Journal on Optimization*, vol. 8, no. 3, pp. 682–706, 1998.
- [36] R. H. Byrd, M. E. Hribar, and J. Nocedal, "An interior point algorithm for large-scale nonlinear programming," *SIAM Journal on Optimization*, vol. 9, no. 4, pp. 877–900, 1999.
- [37] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [38] R. K. Mantiuk, D. Hammou, and P. Hanji, "Hdr-vdp-3: A multi-metric for predicting image differences, quality and contrast distortions in high dynamic range and regular content," *arXiv preprint arXiv:2304.13625*, 2023.
- [39] P. Andersson, J. Nilsson, P. Shirley, and T. Akenine-Möller, "Visualizing errors in rendered high dynamic range images," in *Eurographics - Short Papers*, Eurographics - European Association for Computer Graphics, 2021, pp. 25–28.

- [40] P. Andersson, J. Nilsson, T. Akenine-Möller, M. Oskarsson, K. Åström, and M. D. Fairchild, “Flip: A difference evaluator for alternating images,” *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 3, no. 2, pp. 15–1, 2020.
- [41] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, “The unreasonable effectiveness of deep features as a perceptual metric,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.
- [42] C. Schied et al., “Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination,” in *Proceedings of High Performance Graphics*, 2017, pp. 1–12.
- [43] H. Dammertz, D. Sewtz, J. Hanika, and H. P. Lensch, “Edge-avoiding a-trous wavelet transform for fast global illumination filtering,” in *Proceedings of the Conference on High Performance Graphics*, 2010, pp. 67–75.
- [44] P. Kozłowski and T. Cheblov, “Relax: A denoiser tailored to work with the restir algorithm,” in *GPU Technology Conference*, 2021.
- [45] D. Zhdan, “Reblur: A hierarchical recurrent denoiser,” *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX*, pp. 823–844, 2021.
- [46] C. R. A. Chaitanya et al., “Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder,” *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, pp. 1–12, 2017.
- [47] A. Appel, “Some techniques for shading machine renderings of solids,” in *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, 1968, pp. 37–45.
- [48] S. D. Roth, “Ray casting for modeling solids,” *Computer graphics and image processing*, vol. 18, no. 2, pp. 109–144, 1982.
- [49] T. Whitted, “An improved illumination model for shaded display,” in *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, 1979, p. 14.
- [50] C. Yuksel, “Point light attenuation without singularity,” in *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks*, 2020, pp. 1–2.
- [51] J. Arvo et al., “Backward ray tracing,” in *Developments in Ray Tracing, Computer Graphics, Proc. of ACM SIGGRAPH 86 Course Notes*, 1986, pp. 259–263.
- [52] E. Veach and L. Guibas, “Bidirectional estimators for light transport,” in *Photo-realistic Rendering Techniques*, Springer, 1995, pp. 145–167.
- [53] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, “Modeling the interaction of light between diffuse surfaces,” *ACM SIGGRAPH computer graphics*, vol. 18, no. 3, pp. 213–222, 1984.

- [54] H. W. Jensen, "Global illumination using photon maps," in *Eurographics workshop on Rendering techniques*, Springer, 1996, pp. 21–30.
- [55] H. W. Jensen and N. J. Christensen, "A practical guide to global illumination using photon maps," *SIGGRAPH 2000 Course Notes CD-ROM*, 2000.
- [56] C.-m. Kang, L. Wang, Y.-n. Xu, and X.-x. Meng, "A survey of photon mapping state-of-the-art research and future challenges," *Frontiers of Information Technology & Electronic Engineering*, vol. 17, no. 3, pp. 185–199, 2016.
- [57] F. Tobler and S. Maierhofer, "Improved illumination estimation for photon maps in architectural scenes," in *WSCG '2006: Full Papers Proceedings: The 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2006*, University of West Bohemia, Plzen, Czech Republic, Jan. 2006, pp. 257–262.
- [58] W. Stürzlinger and R. Bastos, "Interactive rendering of globally illuminated glossy scenes," in *Eurographics Workshop on Rendering Techniques*, Springer, 1997, pp. 93–102.
- [59] F. Suykens and Y. D. Willems, "Density control for photon maps," in *Eurographics Workshop on Rendering Techniques*, Springer, 2000, pp. 23–34.
- [60] T. Hachisuka, S. Ogaki, and H. W. Jensen, "Progressive photon mapping," in *ACM SIGGRAPH Asia 2008 papers*, 2008, pp. 1–8.
- [61] B. Spencer and M. W. Jones, "Into the blue: Better caustics through photon relaxation," in *Computer Graphics Forum*, Wiley Online Library, vol. 28, 2009, pp. 319–328.
- [62] T. Zeltner, I. Georgiev, and W. Jakob, "Specular manifold sampling for rendering high-frequency caustics and glints," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 149–1, 2020.
- [63] T. E. Booth, "Unbiased monte carlo estimation of the reciprocal of an integral," *Nuclear science and engineering*, vol. 156, no. 3, pp. 403–407, 2007.
- [64] W. Jakob and S. Marschner, "Manifold exploration: A markov chain monte carlo technique for rendering scenes with difficult specular transport," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–13, 2012.
- [65] X. Xu, L. Wang, and B. Wang, "Efficient caustics rendering via spatial and temporal path reuse," in *Computer Graphics Forum*, Wiley Online Library, vol. 42, 2023, e14975.
- [66] Z. Fan, P. Hong, J. Guo, C. Zou, Y. Guo, and L.-Q. Yan, "Manifold path guiding for importance sampling specular chains," *ACM Transactions on Graphics (TOG)*, vol. 42, no. 6, pp. 1–14, 2023.

- [67] A. Granizo-Hidalgo and N. Holzschuch, “Interactive rendering of caustics using dimension reduction for manifold next-event estimation,” *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 7, no. 1, pp. 1–16, 2024.
- [68] M.-T. Chao, “A general purpose unequal probability sampling plan,” *Biometrika*, vol. 69, no. 3, pp. 653–656, 1982.
- [69] J. F. Talbot, D. Cline, and P. Egbert, “Importance resampling for global illumination,” in *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*, ser. EGSR ’05, Konstanz, Germany: Eurographics Association, 2005, pp. 139–146.
- [70] B. Bitterli, C. Wyman, M. Pharr, P. Shirley, A. Lefohn, and W. Jarosz, “Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting,” *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 148–1, 2020.
- [71] Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and J. Pantaleoni, “Restir gi: Path resampling for real-time path tracing,” in *Computer Graphics Forum*, Wiley Online Library, vol. 40, 2021, pp. 17–29.
- [72] M. Kettunen, M. Manzi, M. Aittala, J. Lehtinen, F. Durand, and M. Zwicker, “Gradient-domain path tracing,” *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, pp. 1–13, 2015.
- [73] D. Lin, M. Kettunen, B. Bitterli, J. Pantaleoni, C. Yuksel, and C. Wyman, “Generalized resampled importance sampling: Foundations of restir,” *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, pp. 1–23, 2022.
- [74] H. Zhang and B. Wang, “World-space spatiotemporal path resampling for path tracing,” in *Computer Graphics Forum*, Wiley Online Library, vol. 42, 2023, e14974.
- [75] P. Moreau and M. Doggett, “Real-time rendering of indirectly visible caustics,” in *17th International Conference on Computer Graphics Theory and Applications, GRAPP 2022; Part of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP 2022*, SciTePress, vol. 1, 2022, pp. 39–48.
- [76] Z. Fan et al., “Specular polynomials,” *ACM Trans. Graph.*, vol. 43, no. 4, Jul. 2024.
- [77] A. Sharif and H.-H. S. Lee, “Total recall: A debugging framework for gpus,” in *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, 2008, pp. 13–20.
- [78] Q. Hou, K. Zhou, and B. Guo, “Debugging gpu stream programs through automatic dataflow recording and visualization,” in *ACM SIGGRAPH Asia 2009 papers*, 2009, pp. 1–11.

- [79] C. Gribble, J. Fisher, D. Eby, E. Quigley, and G. Ludwig, “Ray tracing visualization toolkit,” in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2012, pp. 71–78.
- [80] B. {van Dyk}, C. Lutteroth, G. Weber, and B. Wünsche, “Using opengl state history for graphics debugging,” English, in *Proceedings of the 21th International Conference on Computer Graphics, Visualisation and Computer Vision (WSCG)*, 2013, pp. 29–38, ISBN: 9788086943756.
- [81] S. Podila and Y. Zhu, “A visualization tool for 3d graphics program comprehension and debugging,” in *2016 IEEE Working Conference on Software Visualization (VISSOFT)*, IEEE, 2016, pp. 111–115.
- [82] A. Marrs, P. Shirley, and I. Wald, *Ray tracing gems ii: Next generation real-time rendering with dxx, vulkan, and optix*, 2021.
- [83] T. Alatalo, “An entity-component model for extensible virtual worlds,” *IEEE Internet Computing*, vol. 15, no. 5, pp. 30–37, 2011.
- [84] T. Dahl, T. Koskela, S. Hickey, and J. Vattus-Anttila, “A virtual world web client utilizing an entity-component model,” in *2013 seventh international conference on next generation mobile apps, services and technologies*, IEEE, 2013, pp. 7–12.
- [85] G. Papagiannakis, M. Kamarianakis, A. Protopsaltis, D. Angelis, and P. Zikas, “Project elements: A computational entity-component-system in a scene-graph pythonic framework, for a neural, geometric computer graphics curriculum,” *arXiv preprint arXiv:2302.07691*, 2023.
- [86] S. Kallweit et al., *The Falcor rendering framework*, Aug. 2022. [Online]. Available: <https://github.com/NVIDIAGameWorks/Falcor>.
- [87] N. Bonneel, D. Coeurjolly, J. Digne, and N. Mellado, “Code replicability in computer graphics,” *ACM Trans. Graph.*, vol. 39, no. 4, Aug. 2020.
- [88] R. S. Overbeck, C. Donner, and R. Ramamoorthi, “Adaptive wavelet rendering,” *ACM Trans. Graph.*, vol. 28, no. 5, p. 140, 2009.
- [89] A. Levin, S. W. Hasinoff, P. Green, F. Durand, and W. T. Freeman, “4d frequency analysis of computational cameras for depth of field extension,” *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3, pp. 1–14, 2009.
- [90] K. Egan, Y.-T. Tseng, N. Holzschuch, F. Durand, and R. Ramamoorthi, “Frequency analysis and sheared reconstruction for rendering motion blur,” in *ACM SIGGRAPH 2009 papers*, 2009, pp. 1–13.
- [91] K. Egan, F. Hecht, F. Durand, and R. Ramamoorthi, “Frequency analysis and sheared filtering for shadow light fields of complex occluders,” *ACM Transactions on Graphics (TOG)*, vol. 30, no. 2, pp. 1–13, 2011.

- [92] L. Belcour, K. Bala, and C. Soler, "A local frequency analysis of light scattering and absorption," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 5, pp. 1–17, 2014.
- [93] J.-L. Starck, E. J. Candès, and D. L. Donoho, "The curvelet transform for image denoising," *IEEE Transactions on image processing*, vol. 11, no. 6, pp. 670–684, 2002.
- [94] V. M. Kamble, P. Parlewar, A. G. Keskar, and K. M. Bhurchandi, "Performance evaluation of wavelet, ridgelet, curvelet and contourlet transforms based techniques for digital image denoising," *Artificial Intelligence Review*, vol. 45, pp. 509–533, 2016.
- [95] R. A. Ansari and K. M. Budhhiraju, "A comparative evaluation of denoising of remotely sensed images using wavelet, curvelet and contourlet transforms," *Journal of the Indian Society of Remote Sensing*, vol. 44, pp. 843–853, 2016.
- [96] Y. Yu, D. Li, L. Wang, W. Liu, K. Zhang, and Y. An, "Image denoising method based on curvelet transform in telemedicine," in *Simulation Tools and Techniques: 12th EAI International Conference, SIMUtools 2020, Guiyang, China, August 28-29, 2020, Proceedings, Part II 12*, Springer, 2021, pp. 679–690.
- [97] A. Halidou, Y. Mohamadou, A. A. A. Ari, and E. J. G. Zacko, "Review of wavelet denoising algorithms," *Multimedia Tools and Applications*, vol. 82, no. 27, pp. 41 539–41 569, 2023.
- [98] R. S. Jebur, M. H. B. M. Zabil, D. A. Hammood, and L. K. Cheng, "A comprehensive review of image denoising in deep learning," *Multimedia Tools and Applications*, pp. 1–19, 2023.
- [99] M. E. O'Neill, "Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation," Harvey Mudd College, Claremont, CA, Tech. Rep. HMC-CS-2014-0905, Sep. 2014.
- [100] K. Napoli, K. Bugeja, S. Spina, M. Magro, and A. De Barro, "Anvil: A tool for visual debugging of rendering pipelines," in *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2022) - GRAPP, INSTICC, SciTePress*, 2022, pp. 196–203, ISBN: 978-989-758-555-5. DOI: 10.5220/0010834200003124.
- [101] K. Napoli, K. Bugeja, S. Spina, and M. Magro, "Spectral transforms for caustic denoising: A comparative analysis for monte carlo rendering," in *Advances in Computer Graphics*, N. Magnenat-Thalmann, J. Kim, B. Sheng, Z. Deng, D. Thalmann, and P. Li, Eds., Cham: Springer Nature Switzerland, 2025, pp. 212–225, ISBN: 978-3-031-81806-6.

- [102] P. S. Heckbert, "Adaptive radiosity textures for bidirectional ray tracing," in *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 1990, pp. 145–154.